NOVEL APPROACHES FOR SOLVING LARGE-SCALE OPTIMIZATION PROBLEMS ON GRAPHS

A Dissertation

by

SVYATOSLAV TRUKHANOV

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2008

Major Subject: Industrial Engineering

NOVEL APPROACHES FOR SOLVING LARGE-SCALE OPTIMIZATION PROBLEMS ON GRAPHS

A Dissertation

by

SVYATOSLAV TRUKHANOV

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Sergiy I. Butenko
Committee Members,	Lewis Ntaimo
	Wilbert E. Wilhelm
	Huafei Yan
Head of Department,	Brett A. Peters

August 2008

Major Subject: Industrial Engineering

ABSTRACT

Novel Approaches for Solving
Large-scale Optimization Problems on Graphs. (August 2008)
Svyatoslav Trukhanov, B.S., Kyiv Taras Shevchenko University;
M.S., Kyiv Taras Shevchenko University
Chair of Advisory Committee: Dr. Sergiy I. Butenko

This dissertation considers a class of closely related NP-hard otpimization problems on graphs that arise in many important applications, including network-based data mining, analysis of the stock market, social networks, coding theory, fault diagnosis, molecular biology, biochemistry and genomics. In particular, the problems of interest include the classical *maximum independent set problem* (MISP) and *maximum clique problem* (MCP), their vertex-weighted vesrions, as well as novel optimization models that can be viewed as practical relaxations of their classical counterparts.

The concept of clique has been a popular instrument in analysis of networks, and is, essentially, an idealized model of a "closely connected group", or a cluster. But, at the same time, the restrictive nature of the definition of clique makes the clique model impractical in many applications. This motivated the development of clique relaxation models that relax different properties of a clique. On the one hand, while still possessing some clique-like properties, clique relaxations are not as "perfect" as cliques; and on the other hand, they do not exhibit the disadvantages associated with a clique. Using clique relaxations allows one to compromise between perfectness and flexibility, between ideality and reality, which is a usual issue that an engineer deals with when applying theoretical knowledge to solve practical problems in industry. The clique relaxation models studied in this dissertation were first proposed in the literature on *social network analysis*, however they have not been well investigated

from a mathematical programming perspective.

This dissertation considers new techniques for solving the MWISP and clique relaxation problems and investigates their effectiveness from theoretical and computational perspectives. The main results obtained in this work include (i) developing a scale-reduction approach for MWISP based on the concept of critical set and comparing it theoretically with other approaches; (ii) obtaining theoretical complexity results for clique relaxation problems; (iii) developing algorithms for solving the clique relaxation problems exactly; (iv) carrying out computational experiments to demonstrate the performance of the proposed approaches, and, finally, (v) applying the obtained theoretical results to several real-life problems.

Dedicated to my parents

ACKNOWLEDGMENTS

I would like to express great thanks to Dr. Sergiy Butenko, my committee chair, for his continuous support, encouragement and patience through my doctoral research studies at Texas A&M University. From the first days he has been a patient advisor, a knowledgeable colleague and a real friend. I feel really happy to have an experience of conducting research under his supervision and would like to express my respect and gratitude to him.

Also, I would like to express my thanks to my committee members, Dr. Lewis Ntaimo, Dr. Wilbert Wilhelm, and Dr. Catherine Yan, for their guidance and support throughout the course of this research.

I would like to thank Dr. Balabhaskar "Baski" Balasundaram, Uanny Brens, Sera Kahruman, Reza Seyedshohadaie, Oleksii Ursulenko for being wonderful colleagues and friends at Texas A&M University. Also, I would like to thank my external colleagues and collaborators, especially Dr. Vladimir Boginski from the University of Florida, Dr. Illya Hicks from Rice University and Dr. Andrew Schaefer from the University of Pittsburgh.

Thanks also go to all the faculty of Industrial and Systems Engineering Department, for making my doctoral studies at Texas A&M University a great experience; Dr. Brett Peters and Dr. Guy Curry, for providing me with several opportunities to teach and for their guidance; friendly staff, especially Lesley Bell, Michele Bork and Judy Meeks for helping me with the administrative issues; and professional technical specialists Dennis Allen, Mark Henry and Mark Hopcus for their assistance with the computational part of my research.

Also, I would like to thank all faculty of Computer Science Department at Kyiv Taras Shevchenko University, especially my undergraduate advisor Dr. Stavrovskiy and my graduate advisor Dr. Koval, for providing me base knowledge required for my doctoral study.

Special thanks is addressed to Mr. Kovalyov and Mrs. Osinskaya, my high school teachers, who taught me the basics of mathematics and information technology and played an important role in my future career decisions.

Finally, thanks to my parents for their encouragement and making me who I am. Last, but not least, I would like to thank my wife for her great patience and love.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

TABLE Page

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

In a non-formal way, a graph or a network is defined by a set of dots (vertices, nodes) and links (edges) between them. Since its first introduction in 1735 by L. Euler in his famous Königsberg Bridges problem [55, 62], the concept of graph has been serving for more than 250 years as a convenient, effective, simple and easily understandable mathematical abstraction for modeling many real-life problems. In practical applications, a vertex of a graph usually represents an entity, and an edge represents the relationship of interest between two entities.

For example, in chemistry a molecule can be naturally considered as a graph, where atoms are vertices and bonds between pairs of atoms are edges. Since each atom has its valency that may be greater than one, a molecule is an example of a *graph with multiedges*, where two vertices may be connected by more than one edge. Geographical map is also a graph with the vertices corresponding to the cities and the edges being the roads between cities. In biology, *gene co-expression networks* are graphs where vertices are genes and an edge exists between two vertices if the corresponding genes are co-expressed with correlation higher than a specified threshold, and a *protein interaction network* is represented by a graph with the proteins as vertices and known interactions between pairs of vertices as edges. These are just two examples of many biological structures that may be modeled as graphs [16, 40].

Many application of graphs are found in industry, such as the *phone call* graph constructed in the following way: each phone number is represented by a vertex, and each call placed during a specified time period defines an edge between the

---

The journal model is Mathematical Programming.

corresponding pair of phone numbers. Experiments with the call graph based on a 12-hour time period in 1997 are described in [3, 121], with the corresponding phone call graph having over 53 million vertices and over 170 million edges. The U.S. stock market was modeled as a graph, named the *stock-market graph* in [25]. Here a vertex represents a stock, and two vertices are connected by an edge if the correlation of price fluctuations for the corresponding pair of stocks calculated over a certain period of time exceeds a specified threshold. Internet can be modeled as a graph at detailed level, where vertices correspond to individual computers or other networking devices that have IP address, and edges correspond to the physical links between such devices, as well as at macro level, where nodes are autonomous systems (usually Internet service providers or large companies and organizations) and an edge represents an entry from the global routing table or is obtained by using traceroute or ping probes. The resulting networks are extremely large, even at the macro level the whole network consists of more than 100000 nodes [9, 45].

Finally, graphs may be used to model various social phenomena. In *social networks* the vertices usually represent people and the edges represent a certain type of relationship between them. The well-known "Erdös Number Project" is an example of a *collaboration network*, where two mathematicians are connected if they have published a paper as co-authors [76]. In another example, Figure 1 shows the 2005 *college football schedule graph*, where the vertices are Division I-A college football teams, and two vertices are connected by an edge if the corresponding teams played each other during the considered season.

When appropriate, various attributes may be assigned to the graph's vertices and edges. Attributes could be different by their nature, e.g., in the college football schedule graph one obvious attribute of vertices is the college football team name, and a possible edge attribute is the final score of the corresponding game. In mathematical

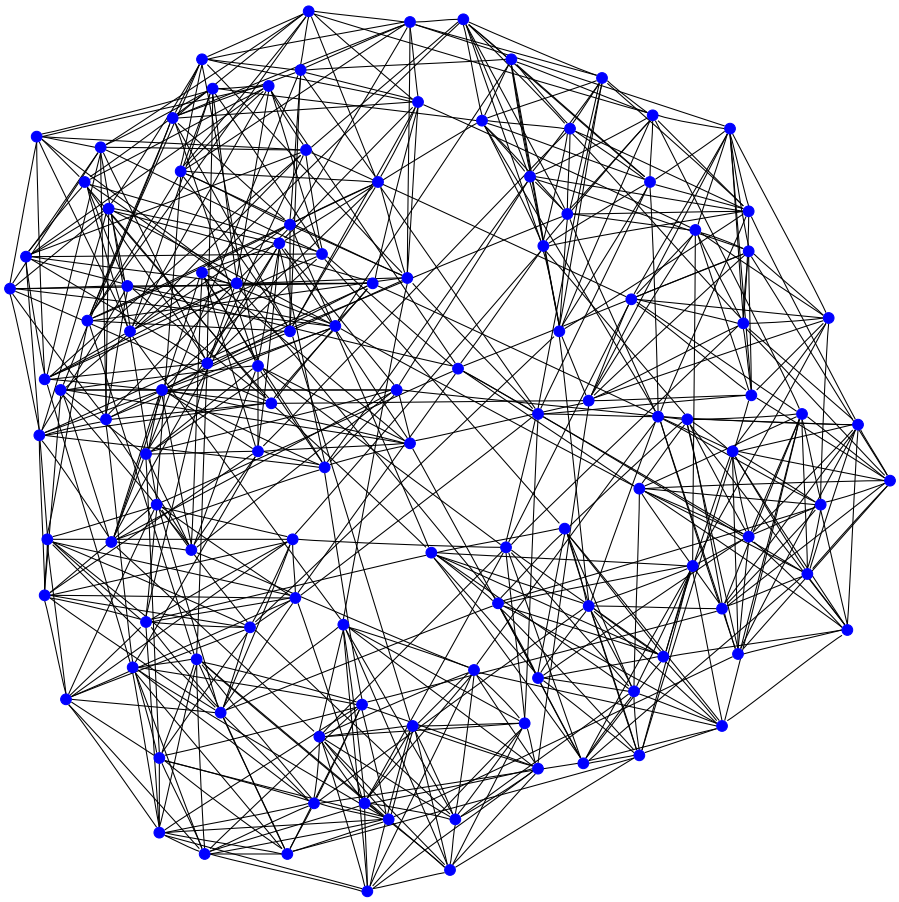


Fig. 1 College football schedule graph for season 2005

programming, only attributes with numerical values are usually considered and called *weights*. Weight functions may be associated with vertices as well as edges.

In different studies of real-life networks, one is required to find the *closely connected groups* in the graph, that are also called *cohesive subgroups* or *clusters*. There are many ways to define the closeness in the group, but the first model that was used in social network analysis utilized the concept of *clique*. A clique is defined as a subset of vertices inducing a subgraph that has all possible edges, i.e., all vertices in a clique are connected to each other. Obviously, a clique represents a "perfectly" connected subgroup, thus it is an ideal model for cohesive subgroups. As an example,

a clique in the aforementioned phone call graph represents a group of people who called each other, thus, most probably, these people are closely connected and share similar interests and preferences. The opposite to the clique structure is an *independent (stable) set*, which is defined as a subset of vertices with no edges in between. These two models are closely related to each other, since a clique in the graph corresponds to an independent set in this graph's compliment, so both models have many common properties, as the problem of finding a clique may be reduced to the problem of finding an independent set in the complement graph and vice versa. In terms of the stock market graph, an independent set represents a *diversified portfolio*, which is one of the key elements of portfolios sought for by the investors.

The problem of finding a clique of the maximum size in the given graph is called the *maximum clique problem*. When a graph has vertex weight, then the problem can be extended to the *maximum weight clique problem*, which requires one to find a clique of the maximum weight in the given graph, where the weight of a clique is defined as the sum of its vertex weights. The complementary problems are the *maximum independent set problem* and the *maximum weight independent set problem*, respectively. These problems have many important applications, including network-based data mining, analysis of the stock market, social networks, coding theory, fault diagnosis, molecular biology, biochemistry and genomics. Moreover, these problems often arise as subproblems of more complicated problems, and many other combinatorial problems may be reduced to these problems. It is well-known that all these four problems are hard to solve, as they belong to the class of NP-hard problems, meaning that there is no effective algorithm to solve the problem in general. However, in practice one still needs to find a way to solve real-life instances of these problems. Sometimes, a provably optimal solution of the problem is required, in which cases an *exact algorithm* is applied to solve the problem. In other cases a non-optimal solution

is acceptable, and a *heuristic algorithm* may be applied that does not guarantee the optimal solution, but provides some "good" solution, usually much faster than the exact algorithm. An efficient way to speed-up the problem solving process is *preprocessing*, that is an algorithm executed before the execution of the main algorithm, aiming to improve the performance of the main algorithm. Also, the properties of graphs arising in real-life problems, such as low *edge-density*, help to solve the problems efficiently. In our research, we consider a new scale-reduction technique for solving the maximum weight independent set problem, which is particularly efficient on sparse graphs.

While the popularity of cliques in network-based studies can be explained by the fact that it represents an ideal "closely connected group", due to its restrictive nature, the clique model becomes impractical in many cases, and has been the subject of the *clique model criticism* in socail networks literature. This motivated the development of the *clique relaxation models* that generalize the clique definition and relax some of the clique requirements. On the one hand, while still possesing some clique-like properties, clique relaxation models are not as "perfect" as cliques; and on the other hand, they do not exhibit the disadvantages that the clique has been criticized for. Using clique relaxations allows one to compromise between perfectness and flexibility, between ideality and reality, which is a usual issue that an engineer deals with when applying theoretical knowledge to solve practical problems in industry. Using clique relaxations in social networks, such as the phone call graph, allows one to find groups of people that may not necessarily be friends, but are still closely connected. Depending of the clique relaxation model used, the way people within a group are connected may also be different, e.g., they may know each other trough the third person, or they may know only some people from the group, etc. The clique relaxation models were first proposed in social network studies rather long time ago [97], but they still

have not been well investigated from a mathematical programming perspective.

In our research, we emphasize three possible relaxations of clique: the *k-clique*, *k-club* and *k-plex* models. Of course, these three clique relaxation models do not cover all possible ways to relax the clique requirements, but they are the most well-known models in social network analysis. First of all, we defined the optimization problems corresponding to the introduced models, provided their mathematical programming formulations and showed that the problems are NP-hard. Next, we concentrated on methods for solving these problems. We developed an exact algorithm for the maximum weight k-plex problem, as well as different scale-reduction techniques for the maximum k-clique, k-club and k-plex problems. Finally, we demonstrated the application of the k-plex model to real life through extensive computational experience.

The remainder of this dissertation is organized as follows. Chapter II provides background on the networks arising in real world problems; introduces the required terms and definitions from graph theory; defines the optimization problems of interest; and provides the relevant literature review. Chapter III concentrates on the scale-reduction technique developed for the MWISP based on the concept of critical weight set. Provided in this chapter theoretical results are used to develop the algorithm. The efficacy of the approach is demonstrated by extensive numerical experiments with large-scale problem instances. Chapter IV investigates the relation between this technique and other scale-reduction approaches for the MWISP. Three more different techniques are considered, the similarity and differences were established and an extension to one of these approaches was proposed.

In Chapter V, we switch from cliques and independent sets to their relaxation models. We define the k-clique, k-club and k-plex formally and consider the relationship between the defined models. Next, the corresponding optimization problems are formulated, their complexity and mathematical programming formulations are

established. Finally, we concentrate our attention on one of the relatively easily solvable case of the introduced problems, the maximum 2-club problem, and develop the corresponding algorithm. Chapter VI is dedicated to the development of the exact algorithm for the maximum weight k-plex problem. The chapter presents the general idea of the algorithm as well as implementation details and discusses possible improvements. The numerical results reported in this chapter allow to evaluate the algorithm's performance and show its superiority to existing approaches.

Finally, Chapter VII demonstrates the application of clique relaxation models in real world, using the instances of market graph. The approach extends the application of combinatorial optimization methods to the market graph, presented in earlier work [27]. Chapter VIII concludes this dissertation and discusses the possible future work.

Some of the results presented Chapters III and V have appeared in [39] and [16], respectively, and papers based on the results of Chapters IV, VI and VII will be submitted for publication.

All figures in this dissertation were generated using GRAPHVIZ software [75] with DOT2TEX converter [57] and all plots were built using PGFPLOTS package [60] for LaTeX.

CHAPTER II

BACKGROUND

This chapter introduces the definitions and notations used throughout this dissertation and provides some background information. Section II.1 discusses graph theory basics. The problem definitions, properties, complexity results and existing solution approaches are discussed in Section II.2. Finally, Section II.3 reviews selected applications of the considered problems in solving real-world optimization problems, points out some issues arising in such applications, and introduces the clique relaxation models that may address the issues raised.

II.1. Graph Theory

Let $G = (V, E)$ be a graph with the vertex (node) set $V = \{1, 2, \ldots, n\}$ and the edge set $E \subseteq V \times V$, where $(i, j) \in E$ if vertices i and j are adjacent. The number of vertices of the graph $n = |V|$ is called the *order* of the graph and the number of edges $m = |E|$ is called the *size* of the graph [53]. Here and later, unless specified explicitly, all considered graphs are assumed to be loopless (edges like (u, u) are not allowed), undirected (edges (u, v) and (v, u) are not distinguishable), and with no multiedges (two vertices may not be connected by more than one edge). The notations $V(G)$ and $E(G)$ denote the vertex and edge sets of graph G, respectively. Given the graph G, its *complement* graph $\bar{G}$ is the graph with the same vertex set that has all possible edges not present in G and no edges present in G. A graph is called *complete* if it contains all possible edges. The complete graph of order n is denoted by K_n. The subgraph induced by $S \subseteq V$ is the graph $G[S] = (S, E \cap (S \times S))$ that has S as its vertex set, and its edge set contains all possible edges from the original graph that connect pairs

of vertices from S. Given a vertex $v \in V$ and a vertex subset $S \subseteq V$, let $V - v$ and $V - S$ denote the graph induced by $V \setminus \{v\}$ and $V \setminus S$, correspondingly. For a set $S \subseteq V$, its *neighborhood* $N(S)$ is the set of all vertices of G that are adjacent to at least one vertex of S. The *closed neighborhood* of S is $N[S] = N(S) \cup S$. If $S = \{s\}$ is a single vertex set, then instead of writing $N(\{s\})$ and $N[\{s\}]$, we will simplify write $N(s)$ and $N[s]$, respectively, and will speak about the node's neighborhood. By $deg_G(v)$ we understand the degree of vertex v in graph G, which is $|N(v)|$. The maximum and minimum degrees of a vertex in graph G are denoted by $\Delta(G)$ and $\delta(G)$, respectively. A *bipartite graph* is a graph whose vertices can be divided into two disjoint sets called *bipartitions* V_1 and V_2, such that every edge connects a vertex in V_1 to a vertex in V_2. The *complete bipartite* graph with bipartitions of sizes p and q is denoted by $K_{p,q}$ and is defined as the bipartite graph with all possible edges between vertices from V_1 and V_2. In particular, the graph $K_{1,n}$ is called a *star*. A *path* in a graph is a sequence of vertices such that each two consecutive vertices in the sequence are connected by an edge. A *cycle* is a path in which the first and the last vertices are the same. Cycle and path on n vertices are denoted by C_n and P_n respectively. A graph is called *connected* if there is a path between every pair of distinct vertices u and v in the graph. A *connected component* of G is defined as a maximal by inclusion connected subgraph of G.

For two vertices u and v from $V(G)$, the *distance* $d_G(u, v)$ between u and v in G is the length of the shortest path between u and v, where the length of a path is measured in the number of edges in this path, i.e., is one less than the number of vertices in the sequence defining the path. The largest distance between any two vertices in G is called the *diameter* of G and is denoted by $d(G)$. For graphs that are not connected, the diameter is assumed to be $+\infty$. For a positive number k, the k-th power of graph G is the graph with the same vertex set $V(G)$ and the edge set given

by $E(G^k) = \{(u, v) : \ d_G(u, v) \leq k\}$.

Let $w \in \mathbb{R}^{|V|}$ ($w \in \mathbb{R}^{|E|}$) be a vector defining weights for the graph's vertices (edges), which will be assumed to be nonnegative, unless specified explicitly. By $w(v)$ or w_v we denote the weight associated with vertex v, and by $w(e)$, w_e and by $w((u, v))$ we denote the weight of edge $e = (u, v)$. $w(S) = \sum_{i \in S} w(i)$ denotes the weight of a vertex or edge subset S, and $w(G) = \sum_{i \in V(G)} w(i)$ denotes the weight of graph G, which is the sum of weights of all graph vertices.

II.2. Independent Sets and Cliques

A subset I of V is called an *independent (stable)* set if the subgraph $G[I]$ induced by I has no edges. A subset C of V is called a *clique* if the subgraph $G[C]$ is a complete graph. An independent set (clique) I is called *maximal* if it could not be extended by adding vertices from $V \setminus I$. A *maximum* independent set (clique) of G is an independent set (clique) of the largest cardinality in G. When speaking about graph vertex subset, the terms *maximal* and *maximum* will be distinguished analogously throughout this work, i.e. *maximal* means maximal by inclusion, while *maximum* means the largest by cardinality or weight. The same is true for the terms *minimal* and *minimum*, i.e., *minimal* by inclusion and *minimum* by cardinality. The cardinality of a maximum independent set is called the *independence (stability)* number, and is denoted by $\alpha(G)$. The cardinality of a maximum clique is called the *clique* number and is denoted by $\omega(G)$. Obviously, if I is an independent set in G then I is a clique in $\bar{G}$ and vice versa, so $\alpha(G) = \omega(\bar{G})$. Therefore, all properties of independent sets in a graph G are also valid for cliques in the complement graph $\bar{G}$.

For a vertex-weighted graph G, a *maximum weight independent (stable)* set is an independent set of the largest weight and the *maximum weight clique* is a clique

with the largest weight in this graph. The weighted clique and stability numbers are redefined correspondingly. Let us note that maximal weight independent set (clique) has no meaning, thus maximality by inclusion will be considered in exactly the same way as in the unweighted case. The maximum weight independent set problem (MWISP) may be formulated as follows: given a graph G with the vertex weights defined by vector w, find an independent set of maximum weight in G [28], and the maximum weight clique problem (MWCP) asks one to find a clique of the maximum weight in the given graph.

The independent set (clique) problem is defined as follows: given a graph G and a positive integer k, does G have an independent set (clique) of cardinality at least k? This is the decision version of an optimization problem, the maximum independent set (clique) problem. From the complexity theory [66, 113], it is a well-known fact that the independent set and clique problems are NP-complete and the maximum independent set and clique problem are NP-hard [66], which means that there are no polynomial-time (also called *efficient*) algorithms for these problems, under assumption that $P \neq NP$. These complexity results are easily extendable to the weighted version of the problems.

The algorithms for any combinatorial optimization problem may be classified by the type of the solution they provide. In such classification, the algorithms that provide exact solutions belong to the class of *exact* algorithms. For the MWISP and MWCP, the exact algorithms ensure finding an optimal solution, but take exponential running time to complete, since the problems are NP-hard. *Heuristic* algorithms or simply *heuristics* unlike the exact approaches, do not guarantee that an optimal solution will be found. In fact, they do not even guarantee that the solution obtained is close to optimal, but they are constructed with the purpose of finding sufficiently good solutions in reasonable time [6,74]. When a solution is obtained, even if it seems

to be of good quality, there is no proof that it is not arbitrarily far away from optimum. Despite of the lack of mathematically strict conclusion about the solution, heuristics are very popular in engineering, since they may provide a practical solution when the exact solution is hard to find or, in case of large-scale real-life problem, virtually impossible to compute due to time limitations. After such solution is obtained, it is evaluated from the engineering point of view and may be used in real life, even without the optimality guarantee.

The *approximation* algorithms are the middle choice between the exact and heuristic ones. Unlike exact algorithms, the approximation algorithms are not guaranteed to find the exact solution, but unlike heuristics, which usually do not provide any performance guarantees, the approximation algorithms come with a bound on how far away the value of the computed solution may be from the optimal one. For $\epsilon > 0$ an ϵ-approximation algorithm has the property that any computed approximate solution cannot have a value less (for the maximization problem) than the factor $(1 - \epsilon)$ times the optimal value. Since approximation algorithms provide solutions that may not be exact, one may want to know what is the complexity of computing an approximate solution. This question has different answer for different problems. While for some problems, such as the *traveling salesman problem* [24, 124] with *Euclidean distance*, there exists a polynomial-time approximation scheme (PTAS), i.e., it is possible to find ϵ-approximation polynomial time algorithm for any $\epsilon > 0$ [10, 132], the maximum weight independent set and clique problems cannot be approximated within any constant factor unless $P = NP$ [11, 83, 114].

Another important class of algorithm are the algorithms that compute bounds for the solution. Of course, any heuristic or approximation algorithm that finds any feasible solution provides a lower bound, but some algorithms may provide the bounds (upper or lower) without even computing a feasible solution. Such algorithms

play important role when one needs to evaluate the optimal value, either in real life problems, or as a subroutine of an exact algorithm, such as a branch-and-bound based algorithm [94, 96].

Next we review some well-known approaches for the maximum weight independent set and clique problems. One class of exact algorithms is based on mathematical programming methods. The advantage of the methods based on mathematical programming formulations is due to the possibility they provide for applying a wide variety of powerful integer, quadratic or non-linear programming techniques for solving the problem. Let x_i be a boolean variable. Then for a vertex set S, the vector $(x_1, x_2, \cdots, x_n)$, where $x_i = 1$ if and only if $i \in S$, is called the characteristic vector of S. The MWISP may be formulated as an integer programming (IP) problem in many possible ways. One of the first and the most well-known formulation is the *edge formulation* provided by Nemhauser and Trotter in 1975 [110] is given by:

$$\begin{aligned} \alpha(G) &= \max \sum_{i \in V} w_i x_i, \\ &\text{s.t} \quad x_i + x_j \leq 1 \qquad \forall (i,j) \in E, \\ &\qquad x_i \in \{0,1\} \qquad \forall i \in V. \end{aligned} \tag{2.1}$$

where the constraints ensure that two adjacent vertices cannot be included in the set together at the same time. Another interesting IP formulation was recently found by Balasundaram [14], when investigating an IP formulation of the maximum k-plex problem:

$$\begin{aligned} \alpha(G) &= \max \sum_{i \in V} w_i x_i, \\ \text{s. t. :} &\quad \sum_{j \in N(i)} x_j \leq deg_G(i)(1 - x_i) \quad \forall\ i \in V, \\ &\quad x_i \in \{0,1\} \qquad\qquad\qquad \forall\ i \in V. \end{aligned} \tag{2.2}$$

The first formulation has as many constraints as the number of edges in the graph, and each constraint involves only two variables, so the corresponding constraint matrix is

large and sparse. The second formulation, in contrast, creates a relatively small but dense constraint matrix.

A wide variety of IP techniques may be applied to such formulations of the MWISP, including the general, classical IP approaches [49, 67, 73, 109, 137, 138], as well as specialized approaches designed specifically for the MWISP. For example, Warrier et al. [133] used a branch-and-price (BP) approach to decompose the original graph into smaller subgraphs, solve the MWISP in such subgraphs (which is much easier than solving the original problem), and use the obtained information to generate columns in a BP framework for the original graph. By ignoring the integrality requirement for the decision variables, the linear programming (LP) relaxation is obtained that provides an upper bound for the problem solution. The LP relaxation is just a linear problem, that may be solved efficiently using the classic LP approaches [22, 47, 51].

Other well-known mathematical programming formulations of the MWISP are quadratic boolean optimization formulations. The advantage of these formulations is the absence of the constraints on the decision variables, except for binary restrictions. One such formulation is studied by Hammer et al. [32, 33]:

$$\alpha(G) = \max_{x_i \in \{0,1\}} \left(\sum_{i \in V} w_i x_i - W \sum_{(i,j) \in E} x_i x_j \right), \tag{2.3}$$

where $W = w(V)$ is the weight of graph G. Given an optimal solution x^* to this problem, the set $I = \{v : x_v^* = 1\}$ is a maximum independent set of G.

In addition, there exist numerous continuous optimization formulations of the maximum independent set problem, one of which was provided in [14]:

$$\alpha(G) = \max_{x \in [0,1]^n} \sum_{i \in V} \frac{x_i}{1 + \sum\limits_{j \in N(i)} x_j}, \tag{2.4}$$

where $N(j) = \{v : (j, v) \in E\}$ is the neighborhood of vertex j. Note that, unlike two previous formulations, this formulation allows one to find the size of the maximum independent set $\alpha(G)$ but not the set itself. However, any global maximum of this formulation corresponds to a subset of vertices inducing a subgraph whose connected components are cliques (the so-called independent union of cliques), and a maximum independent set can be obtained by taking one vertex from each such clique. More information on non-linear programming approaches for the MWISP may be found in [36, 68, 108, 115].

All known exact combinatorial algorithms for solving the problems of interest have exponential time complexity. Many such algorithms utilize branch-and-bound ideas. Based on a simple heuristic strategy, Carraghan and Pardalos [41] designed one of the best-known exact algorithms for the maximum clique problem that performs particularly well on sparse graphs. Using different idea, that is also heuristic, Östergård developed the algorithm for the maximum weight clique problem, which is considered the state of the art at the moment [111, 112]. The benefit of the combinatorial algorithms is their higher performance compared to the IP approach, due to narrow algorithm specifications.

A wide variety of heuristic techniques was successfully employed in order to solve the maximum weight independent set and clique problems [118]. Even simple local search heuristics [2], may be applied to the problem [21]. The tabu search [69–71] was applied to problems of interest in [63, 128]; Feo and Resende [58, 59] proposed the Greedy Randomized Adaptive Search Procedure (GRASP) for the maximum independent set problem; simulated annealing introduced in [1,92] is also applicable to these problems [29, 30]. Finally, neural networks [84, 143] and genetic algorithms [72] were successfully used for solving the maximum independent set problem in [87, 88] and [31, 82, 101], correspondingly. In addition to the classic approaches, there are

many special heuristics developed for the problem, such as a relatively new but very promising Global Equilibrium Search by Shylo et al. [116, 126].

The importance and popularity of the MWISP, is the reason why there are so many approaches developed for this problem, but at the same time, the problem also serves as a good challenging benchmark for evaluate the performance of general-purpose approaches.

The methods described above may be applied to all instances of the MWISP and, due to the NP-hardness of the problem, they have exponential time complexity. From the practical point of view this means excessively long running time in general. However, in many cases, the graphs of interest have some properties that may be exploited and utilized in solving the maximum independent set problem. Hence, many studies of the MWISP are restricted to some special classes of graphs. In some cases such studies include strict theoretical results and provide exact algorithms (as an example, Minty showed that the maximum independent set problem can be solved in polynomial time on a claw-free graph and provided the algorithm for finding the maximum independent set that is applicable to this graph class only [106]). Other studies provide theoretical results without practical algorithms (e.g., Hunt et al. [85] obtained the result of existence of a polynomial-time approximation scheme for the maximum independent set problem in unit disk graphs, but presented no algorithm that can utilize this fact to solve the problem faster). And, finally, many results have been justified by numerical study only, without any theoretical proof (e.g., based on numerical experiments, it is known that Carraghan-Pardalos [41] and Östergård [111, 112] algorithms perform extremely well with many instances, but there are no additional theoretical results related to the complexity of the algorithm, except that the algorithms have exponential complexity in general).

A very useful technique in solving the MWISP is preprocessing. Preprocessing

may be defined as an optional additional step that is performed before the main algorithm or one of its iterations is executed. Preprocessing could either reduce the size of the input instance (this is sometimes referred to as *kernelization* [43, 46]), or speed up the main algorithm [111]. Such techniques are usually very specific to a particular instance and algorithm. Preprocessing procedures exist both for combinatorial algorithms [39, 43, 46] and for approaches based on mathematical programming formulations [32, 33, 110].

II.3. Real Life Networks and Clique Relaxations*

Study of biological networks and other complex networks such as the Internet and the world wide web, introduced earlier in Chapter I, have received special attention from scientists because of their interesting properties and the information they hold. In this respect, the concept of *scale-free networks* [8, 18] is a recent development. It has been observed that the degree distributions of a large number of such complex networks follow a power law. As a consequence, average degree is no longer representative and a majority of the nodes have few neighbors, while a smaller number of nodes have very high degrees.

The principle of preferential attachment, which suggests that the new nodes have a higher probability to link to nodes that already have a high degree, is used to explain the power-law degree distribution of such scale-free graphs. In addition, these networks are also hierarchical in the sense that they can be partitioned into a collection of functional modules. Analysis of several biological networks provides strong evidence that biological networks are both scale-free and modular. Identifying

*Parts of this section are reprinted with permission from Balasundaram, B., Butenko, S., Trukhanov, S.: Novel approaches for analyzing biological networks. Journal of Combinatorial Optimization **10**(1), 23–39 (2005) © Springer.

large clusters or functional modules in biological networks can aid different objectives depending on the nature of these networks. Clique models have been most popular in this area as they represent *tight clusters* in a network. Cliques have been used to cluster *gene co-expression networks* [90,119]. Cliques and high density subgraphs have also been used to cluster *protein interaction networks* in [65, 129]. However, clique models could be overly restrictive in describing clusters in such networks. Graph theoretic clique relaxations that are used in social network analysis for identifying *cohesive subgroups* can provide interesting insights into these networks and provide more information than what is revealed by cliques. Relaxing the restrictions imposed by clique models could reveal new protein interactions. In particular, structures where interactions of proteins occur through a central protein, which are likely to be found in similar biological processes, can be identified [13] by the models suggested in this paper.

Besides biological networks, cohesive subgroups can be used to cluster airline networks where reachability is a critical issue. An important classical application of cohesive subgroups is the study of terrorist and other criminal networks [12, 42, 52]. More recently, these models have been used to study web graphs in Internet research [130] to facilitate organization and faster retrieval of information from the web. These approaches have also been used in clustering wireless networks [93] and for other graph based data mining applications [48, 61, 134].

Among other applications, cliques are often used to represent *clusters* of similar elements. For example, in social networks, a clique represents a group of people such that any two of them have a certain kind of relationship (friendship, acquaintance, etc.) with each other [107]. In fact, some of the earliest works addressing the concept of cliques and methods of their detection were motivated by applications in sociometry [78, 97, 98]. Social network analysis requires three properties in a cohesive

subgroup model: *familiarity*, *reachability* and *robustness*, that translate to degree, distance/diameter and connectivity in graph theory. Clique is ideal with respect to these three properties: it provides the maximum possible familiarity (degree) among clique members, it has the smallest possible pairwise reachability (distance) between members, as well as, the smallest possible diameter of the whole graph, finally, clique has the maximum possible robustness (connectivity).

The clustering problems studied in this dissertation deal with *relaxations* of the idea of a clique, in which, for any two vertices, the requirement of their connectedness is replaced with a less tight condition on the distance between them. We first state the corresponding definitions of k-clique, k-clan and k-club as they originally appeared in the literature. Following which, we will point out some drawbacks in these definitions and modify them according to standard definitions of similar concepts in graph theory. It is not surprising that the clustering concepts of interest first appeared in studying cohesive subgroups in social networks, where the vertices correspond to *actors* in a social network and an edge indicates a relationship between two actors [135].

Luce [97] defines an *k-clique* of G as a subset of vertices $C \subseteq V$ such that for all $u, v \in C : \ d_G(u, v) \leq k$ and this subset is maximal by inclusion. In other words, an k-clique C is a set of vertices in which any two vertices are a distance of at most k from each other in G, and no other vertex in the graph is of distance k or less from every other vertex in C. Thus, if two vertices $u, v \in V$ belong to an k-clique C, then $d_G(u, v) \leq k$, however this does not imply that $d_{G(C)}(u, v) \leq k$. Hence, the concept of k-clique lacks the requirement of *tightness* in the group corresponding to vertices of a k-clique, while such a requirement is essential to applications in social networks. This observation motivated Alba [7] to introduce the concept of a *sociometric clique*, which was later renamed to *k-clan* by Mokken [107]. An k-clique C is called an *k-clan* if the diameter of the induced subgraph $G(C)$ is no more than k. Finally, Mokken [107]

defines an *k-club* to be a maximal (by inclusion) subset of vertices, $D \subseteq V$ such that the diameter of the induced subgraph $G(D)$ is at most k. A study of relations between cliques, clans and clubs in a graph can be found in [107].

Even though the concepts just defined are used quite extensively in social networks analysis and are even covered in standard textbooks (see, e.g., [135]), their definitions have some deficiencies from the mathematical viewpoint. One considerable drawback of the k-clan definition is that for some graphs an k-clan may not exist. This point is illustrated in Figure 2, which shows a graph with two 2-cliques $\{1, 2, 3, 4, 5, 6, 7\}$ and $\{1, 2, 3, 5, 6, 7, 8\}$, neither of which is a 2-clan.

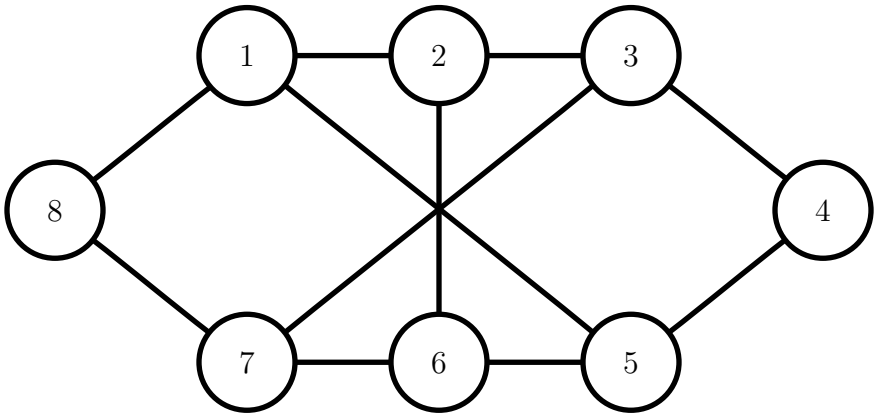


Fig. 2 A graph with no 2-clans

Some other difficulties arise from the requirement of maximality (by inclusion) in all three definitions. In particular, this requirement makes checking whether a given subset of vertices is an k-club a nontrivial matter. Indeed, to check that C is a k-clique, it suffices to show that there is no vertex outside C that could be added to C without violating the requirement that all pairwise distances between vertices do not exceed k. A similar criterion would not work for k-club, however, since in this case the maximality by inclusion is not equivalent to nonexistence of one vertex that could increase the size of the k-club [107].

Taking into account that the above definitions of 1-clique, 1-clan and 1-club all correspond to the standard definition of a *maximal* clique, we proposed to modify the definitions of k-clique and k-club accordingly [16]. Namely, by a k-clique of graph $G = (V, E)$ we will mean a subset of vertices C, such that for any $u, v \in C$: $d_G(u, v) \leq k$. Similarly, by an k-club we will understand a subset of vertices D such that $diam(G(D)) \leq k$. A similar definition of k-clan becomes redundant. The example in Figure 2 suggests the impracticality of such a concept, so we do not consider k-clans in the further discussion.

Finally, a degree-based relaxation, known as a *k-plex*, was defined by Seidman and Foster [125]. A k-plex is a subset of vertices S such that, for each vertex $v \in S$, the degree of v in the induced subgraph $deg_{G[S]}(v) \geq |S| - k$. If $k = 1$, then the k-plex, as previous relaxations, corresponds to the standard definition of a clique.

To demonstrate the advantage of the clique relaxation models over the regular clique, consider as an example, the college football schedule graph, shown in Figure 1. Even though this graph has the maximum clique of order 9, that does not provide much useful information about the graph structure. On the other hand, Figure 3 presents the same graph, with vertices being grouped according to the maximal 4-plexes found in the graph. In such representation, one may easily observe the college football schedule structure, that exactly corresponds to the conferences structure: all teams are divided into 11 conferences of 8 to 12 teams each, and there are 4 independent teams that do not belong to any conference.

An interesting question about cohesive subgroups is, what happens if one or more members leave the subgroup? Will the subgroup preserve its structure in such case? For the clique, independent set, k-clique and k-plex this is true, bot not for the k-club. Strictly speaking, the graph property Π is called *hereditary on induced subgraphs*, i.e., if G is a graph with property Π, then deletion of any nodes does not

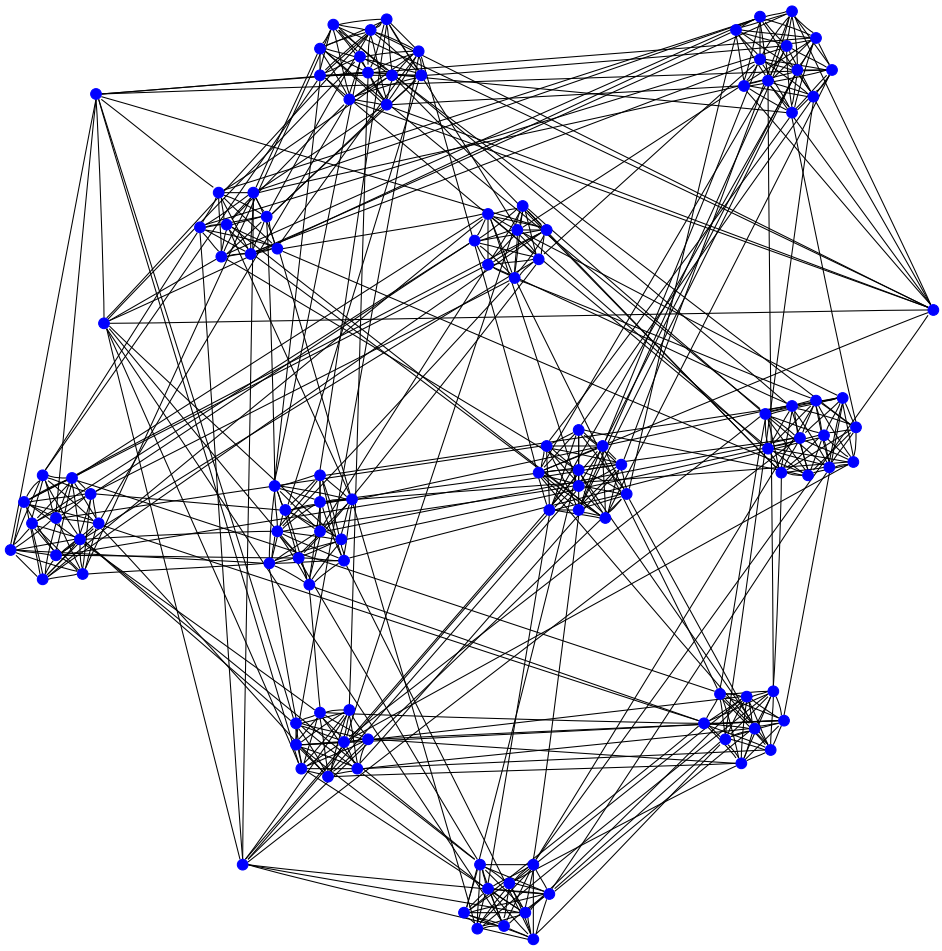


Fig. 3 College football schedule graph clustered on 4-plexes

produce a graph violating Π. Property Π is *nontrivial* if it is true for a single node graph and is not satisfied by all the graphs. Finally, a property is *interesting* if there are arbitrarily large graphs satisfying Π. The maximum Π problem asks to find the maximum (or maximum weight) induced subgraph that does not violate property Π. Yannakakis [140] obtained a general complexity result for such properties Π, that can be restated under our definitions as follows.

Theorem 1 (Yannakakis, 1978)**.** *The maximum* Π *problem for nontrivial, interesting graph properties that are hereditary on induced subgraphs is NP-hard.*

Clearly, in social network studies one is interested with nontrivial and interesting properties only, so it looks like being hereditary on induced subgraphs is the source of the hardness of the problem. But one should not think that if a property is not hereditary on induced subgraphs, then the problem becomes easier. In fact, the k-club is not hereditary on induced subgraphs, but even verifying maximality by inclusion (which is usually easier than finding the maximum subgraph) is a nontrivial problem.

Even though not stated explicitly, the cohesive subgroup are intended to be connected. Cliques and k-clubs are always connected, but k-cliques and k-plexes may not be. A graph consisting of two disjoint cliques of order $k-1$ should be considered as two separate cliques rather than as a k-plex. Again, a restricted version of the problem, *the maximum connected* Π *problem*, may be formulated as follows: given a graph, find a maximum *connected* subgraph with the property Π. The effect of connectivity was considered again by Yannakakis [141] and the corresponding result may be stated as follows.

Theorem 2 (Yannakakis, 1979)**.** *The maximum connected* Π *problem for graph properties that are hereditary on induced (connected) graphs, nontrivial and interesting on connected graphs is NP-hard.*

Indeed, these two complexity results do not cover all possible definitions of the cohesive subgroups, but they give the idea that problems of finding cohesive subgroups are not easy to solve in general.

All clique relaxation models considered above were based on relaxation requirements for vertex properties of a clique. There are also some models based on relaxing edge properties for clique. Clique has the maximum possible number of edges between its members, which is equal to $n(n-1)/2$ for a clique of order n. For a given $0 \geq \gamma \geq 1$, the authors of [4] defined the γ-clique or *quasi-clique* as a vertex set $S \subseteq V(G)$, such

that the graph $G[S]$ is connected and has *edge density* at least γ, where the edge density is defined in an obvious way as $|E(G[S])|/(|S|(|S|-1)/2)$. There is some confusion with terminology, since other authors define quasi-clique based on vertex degrees. For example, [117] defines a γ-*complete* graph as a connected graph G such that every vertex in the graph has a degree at least $\gamma(|V(G)|-1)$. They define a γ-quasi-clique as a maximal by inclusion γ-complete subgraph of G. For $\gamma = 1$ both definitions for quasi-clique are just the regular clique. Both variants of defined quasi-clique models are widely used in biological and social network analysis along with many other models that are out of the scope of this dissertation [44, 56, 144].

Even though the concept of cohesive subgroups is borrowed from social network analysis, these ideas are applicable to any network, and finding these cohesive subgroups can reveal several important structural aspects of the networks. Despite a number of important practical applications, the combinatorial optimization problems concerned with finding large k-cliques, k-clubs and k-plexes have not been well studied analytically or computationally. In fact, little has been known about the complexity aspects of such problems and mathematical programming approaches to these problems are in their infancy.

CHAPTER III

SCALE REDUCTION APPROACH FOR THE MAXIMUM WEIGHT INDEPENDENT SET PROBLEM*

In this chapter, we develop a method that utilizes the polynomially solvable critical weight independent set problem for solving the maximum weight independent set problem on graphs with a nonempty critical weight independent set. Section III.1 provides definitions and background of critical weighted sets and critical weight independent sets. Next, in Section III.2, we establish the relationship between critical independent and maximum independent sets that allows us to develop the algorithm in Section III.3. The effectiveness of the proposed approach on large graphs with large independence number is demonstrated through extensive numerical experiments in Section III.4.

III.1. Critical and Critical Independent Sets

Let $G = (V, E)$ be a simple undirected graph with vertex weights $w : V \to \mathbb{R}_+$. Zhang [142] introduced the critical sets and critical independent sets as follows:

A vertex set $U_c \subseteq V$ is called *critical* if

$$\mu_c(G) = |U_c| - |N(U_c)| = \max\{|U| - |N(U)| : \ U \subseteq V\}. \tag{3.1}$$

The number μ_c is called the *critical number* of G. An independent set $I_c \subseteq V$ is called

*Parts of this chapter are reprinted with permission from Butenko, S., Trukhanov S.: Using critical sets for the maximum independent set problem solving. Operations Research Letters **35**(4), 519–524 (2007) © Elsevier B.V.

critical if

$$\alpha_c(G) = |I_c| - |N(I_c)| = \max\{|I| - |N(I)| : \ I \text{ is an independent set of } G\}. \quad (3.2)$$

The corresponding number $\alpha_c(G)$ is called the *critical independence number* of G. Later Ageev [5] made a generalization of these definitions to graphs with vertex weights. A vertex set $U_c \subseteq V$ is called *critical weighted* if

$$\mu_c(G) = w(U_c) - w(N(U_c)) = \max\{w(U) - w(N(U)) : \ U \subseteq V\}. \quad (3.3)$$

$I_c \subseteq V$ is a *critical weight* independent set if

$$\alpha_c(G) = w(I_c) - w(N(I_c)) = \max\{w(I) - w(N(I)) : \ I \text{ is an independent set of } G\}. \quad (3.4)$$

The main result provided by Ageev [5] is that the problems of finding a critical weighted and critical weight independent set are polynomially solvable, moreover $\alpha_c(G) = \mu_c(G)$. Obviously, $\alpha_c(G) \leq \mu_c(G)$. Let U_c be a critical weighted set in G, so $\mu_c(G) = w(U_c) - w(N(U_c))$. Let $A \subseteq U_c$ be a set of non-isolated vertices of U_c in $G[U_c]$. Then $A \subseteq N(U_c)$ and hence $A \subseteq U_c \cap N(U_c)$, but in such case

$$\begin{aligned} & w(U_c \setminus A) - w(N(U_c \setminus A)) \\ \geq\ & w(U_c) - w(A) - (w(N(U_c)) - w(A)) \\ =\ & w(U_c) - w(N(U_c)) = \mu_c(G). \end{aligned}$$

On the other hand, U_c is a critical weight set, so $w(U_c \setminus A) - w(N(U_c \setminus A)) = \mu_c(G)$, moreover $V \setminus A$ is independent by construction, so $I_c = U_c \setminus A$ is a critical weight independent set of G and $\alpha_c(G) = \mu_c(G)$. This proof also provides a way to find a critical weight independent set from a known critical weight set: just take all isolated vertices in $G[U_c]$ and the resulting set will be a critical weight independent set of G.

To show that the problem of finding a critical weighted set is solvable in polyno-

mial time, first consider its IP formulation:

$$\begin{array}{rcll} \mu_c(G) & = & \max \sum\limits_{v \in V} w_v x_v - \sum\limits_{v \in V} w_v y_v, & \\ & \text{s.t.} & y_v \geq x_u & \forall (u,v) \in E, \\ & & x_v, y_v \in \{0,1\} & \forall v \in V. \end{array} \tag{3.5}$$

The formulation means that we try to minimize the difference between the weights of two vertex subsets such that the vertex from second subset is chosen whenever at least one of its neighbor is chosen. Ageev has shown that the optimal solution parts x^* and y^* correspond to the critical weighted set and its neighborhood, respectively. Next, let us make the substitution $z_v = 1 - y_v \ \forall v \in V$, then the problem (3.5) transforms to:

$$\begin{array}{rcll} \mu_c(G) & = & |V| + \max \sum\limits_{v \in V} w_v x_v + \sum\limits_{v \in V} w_v z_v, & \\ & \text{s.t.} & y_v + z_u \leq 1 & \forall (u,v) \in E, \\ & & x_v, z_v \in \{0,1\} & \forall v \in V, \end{array} \tag{3.6}$$

which is, up to the constant additive $|V|$, an IP formulation for an instance of the maximum weight independent set problem on $B(G)$, the so-called *bidual* graph of G, whose vertex set is defined by $V(B(G)) = V \cup V'$, where V' is a copy of vertex set V, and the edge set defined as $E(B(G)) = \{(u,v'),(u',v) : (u,v) \in E(G)\}$. Simply speaking, graph $B(G)$ is a bipartite graph, each of the two partitions of which is a copy of vertices of the original graph, and the edge between vertices from different partitions exists if and only if the corresponding vertices in the original graph are connected. The problem of finding a maximum weight independent set on a bipartite graph is known to be polynomially solvable [66] by reduction to the maximum flow (minimum cut) problem on graph $B'(G)$, which is a directed graph with edge weights. $B'(G)$ is obtained from $B(G)$ by adding artificial vertices s (source) and s' (sink), connecting s to all vertices from V, connecting all vertices from V' to s', and directing all edges

of $B(G)$ from V to V'. The weights on edges (upper limit for flow) are equal to w_v for edges $\{(s, v) : v \in V\}$ and $\{(v', s') : v' \in V'\}$, edges from V to V' have no limits (i.e., the weights are equal to $+\infty$). Vertices that are not adjacent to the edges from the minimum cut in $B'(G)$ create a maximum weight independent set in $B(G)$ [77] and thus, provide the way to find a critical weighted set in the original graph G. This way is not necessarily the best one for finding a critical weighted set, but it establishes the complexity result and plays an important role in further theoretical research. Figure 4 shows the original graph and illustrates the corresponding network flow problem on $B'(G)$.

III.2. Relation Between Critical Independent and Maximum Independent Sets

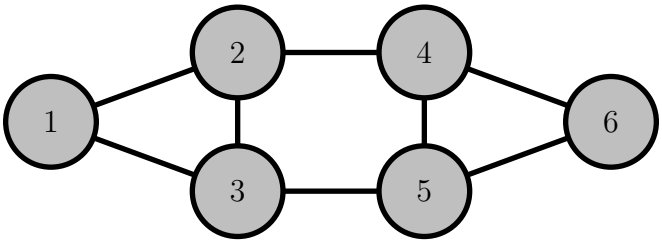
This section establishes the results relating the critical weighted independent set problem to the maximum weight independent set problem. In particular, the main theorem states that any critical weighted independent set is a subset of a maximum weight independent set. This fact is then used to develop a scale-reduction procedure for the maximum independent set problem in graphs with a nonempty critical independent set.

Lemma 3. *If I_c is a critical weighted independent set and a maximal independent set, then I_c is a maximum weight independent set.*

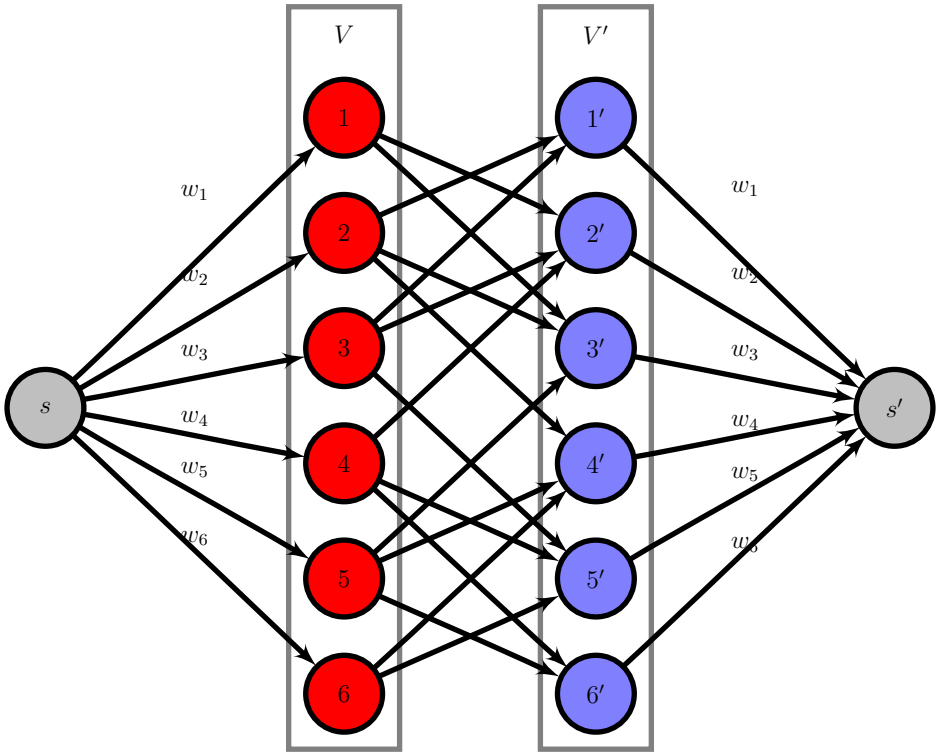
Proof. Since I_c is a maximal independent set, we have $N(I_c) = V \setminus I_c$. Assume that there exists an independent set I with weight $w(I) > w(I_c)$. Then $w(V \setminus I) < w(V \setminus I_c)$ and

$$w(I) - w(N(I)) \geq w(I) - w(V \setminus I) > w(I_c) - w(V \setminus I_c),$$

which contradicts to the fact that I_c is a critical weighted independent set. □



(a) Original graph G



(b) Graph $B'(G)$ for the network flow problem

Fig. 4 Critical set to network flow reduction

Corollary 1. *If I_c is a critical weighted independent set and a maximal independent set, then $w(I_c) \geq w(V)/2$.*

Proof. Follows from the non-negativity of $w(I_c) - w(N(I_c))$. □

Theorem 4. *If I_c is a critical weighted independent set, then there exists a maximum weight independent set I, such that $I_c \subseteq I$.*

Proof. Let J be a maximum weight independent set in G, and I_c be a critical weighted independent set. Put

$$I = (J \cup I_c) \setminus N(I_c).$$

Then I is an independent set, and $I_c \subseteq I$. To prove that I is a maximum weight independent set, it suffices to show that $w(I) \geq w(J)$. Assume that $w(I) < w(J)$, then $w(J \setminus I) > w(I \setminus J)$. Since $I \setminus J = I_c \setminus J$ and $J \setminus I = N(I_c) \bigcap J$, we obtain

$$w(N(I_c) \cap J) > w(I_c \setminus J).$$

Using the last inequality and the inequality

$$w(N(I_c)) \geq w(N(I_c) \cap J) + w(N(I_c \cap J)),$$

we have

$$\begin{aligned} w(I_c) - w(N(I_c)) &= w(I_c \setminus J) + w(I_c \cap J) - w(N(I_c)) \\ &< w(N(I_c) \cap J) + w(I_c \cap J) - w(N(I_c) \cap J) - w(N(I_c \cap J)) \\ &= w(I_c \cap J) - w(N(I_c \cap J)). \end{aligned}$$

We obtain a contradiction with the fact that I_c is a critical weighted independent set. □

Theorem 4 states that a nonempty critical weighted set is always a part of a maximum weight independent set, therefore it can be used in order to reduce the

number of vertices to be analyzed when solving the maximum weight independent set problem. The following two lemmas provide some elementary properties of critical weighted sets and critical weighted independent sets in a graph that will be used in the reduction algorithm.

Lemma 5. *Let U be a critical weighted set of the simple undirected graph $G = (V, E)$. Then $U' = U \cup W$, where $W = V \setminus (U \cup N(U))$ is also a critical weighted set of G and $U' \cup N(U') = V$.*

Proof. By the definition of W, $U \cap W = \emptyset$ and $N(U) \cap W = \emptyset$, hence

$$N(W) \subseteq V \setminus U \subseteq W \cup N(U).$$

Since $U' = U \cup W$, we have

$$N(U') = N(U) \cup N(W) \subseteq N(U) \cup W \cup N(U) = W \cup N(U).$$

Thus,

$$\begin{aligned} w(U') - w(N(U')) &= w(U) + w(W) - w(N(U')) \\ &\geq w(U) + w(W) - w(W \cup N(U)) \\ &= w(U) - w(N(U)), \end{aligned}$$

so U' is also a critical weighted set of G.

Note that $U' \cup N(U') = U \cup W \cup N(U) \cup N(W) = V$. □

Lemma 6. *Let U be a critical weighted set of $G = (V, E)$, such that $U \cup N(U) = V$, $I \subseteq U$ be a critical weighted independent set obtained from U by taking isolated vertices of $G(U)$. Then*

$$I = U \setminus N(U),\ N(I) = N(U) \setminus U,$$

and

$$V \setminus (I \cup N(I)) = U \cap N(U) = U \setminus I.$$

Proof. We first show that $I = U \setminus N(U)$. Consider $v \in I$. By the definition of I, $v \in U$. Assume that $v \in N(U)$, then there exists $w \in U, w \neq v : (v, w) \in E$, so v is not isolated in $G(U)$, therefore $v \notin I$. So, $I \subseteq U \setminus N(U)$. On the other hand, if $v \in U \setminus N(U)$ then $v \in U$ and is isolated in $G(U)$, so $v \in I$. Thus, $U \setminus N(U) \subseteq I$, so $I = U \setminus N(U)$.

Next we show that $N(I) = N(U) \setminus U$. Note that since $U \cup N(U) = V$ and $N(I) \cap U = \emptyset$, we have $N(I) \subseteq N(U) \setminus U$. Recall that from [5] we know that $w(I) - w(N(I)) = w(U) - w(N(U))$, but

$$w(U) - w(N(U)) = w(U \setminus N(U)) + w(U \cap N(U)) - w(N(U) \setminus U) - w(U \cap N(U))$$

$$= w(I) - w(N(U) \setminus U).$$

So, $w(N(I)) = w(N(U) \setminus U)$ and, since $N(I) \subseteq N(U) \setminus U$, we have

$$N(I) = N(U) \setminus U.$$

Finally,

$$\begin{aligned} V \setminus (I \cup N(I)) &= (U \cup N(U)) \setminus ((U \setminus N(U)) \cup (N(U) \setminus U)) \\ &= (U \cup N(U)) \setminus (U \triangle N(U)) \\ &= U \cap N(U) \\ &= U \setminus I, \end{aligned}$$

where $\triangle$ denotes the symmetric difference. □

III.3. Scale-reduction Algorithm

Theorem 4 allows one to reduce the size of the maximum weight independent set problem and could be used as a preprocessing step before the main algorithm is

applied. This idea is implemented in Algorithm 1:

Algorithm 1 Critical set based scale-reduction algorithm

```
1: procedure CriticalReduction(G)
2:     I ← ∅
3:     repeat
4:         U_c ← Critical(G)
5:         I_c ← U_c \ N(U_c)
6:         I ← I ∪ I_c
7:         V ← V \ I_c \ N(I_c)
8:         G ← G[V]
9:     until I_c = ∅
10:    G_r ← G
11:    return I ∪ MIS(G_r)
12: end procedure
```

1. Initialize $I = \emptyset$.

2. Compute a critical set U_c of G. To find a critical set U_c in the unweighted case, first use a reduction of the critical set problem to the maximum matching problem in a bipartite graph as proposed by Ageev [5] and then apply a standard algorithm for computing a maximum matching in bipartite graph in $O(|E||V|)$ time (see, e.g., the proof of König's theorem in [49]). In the general case, the critical weighted set problem may be reduced to the selection problem [5], which is equivalent to finding a minimum cut (or maximum flow) in a bipartite graph [17, 122].

3. Compute a critical weight independent set of G by putting $I_c = U_c \setminus N(U_c)$, which is equivalent to removing all non-isolated vertices from $G(U_c)$ [5, 39].

4. If $I_c \neq \emptyset$, put $I = I \cup I_c$, $V = V \setminus (I_c \cup N(I_c))$, $G = G(V)$ and go to step 2.

5. Denote the remaining graph G by $G_r = (V_r, E_r)$ and output G_r.

6. Find a maximum weight independent set in G_r. Any exact algorithm (e.g., [41, 111]) may be used for this purpose.

7. The maximum weight independent set of G will be a union of I and the maximum weight independent set of G_r.

III.4. Numerical Experiments

We tested the critical weighted set approach to the maximum weight independent set problem on graphs $G = (V, E)$ with unit vertex weight and $\alpha(G) > |V|/2$, since in this case a critical independent set I_c is guaranteed to be nonempty. Note that it is easy to prove that the maximum independent set problem remains NP-hard even if restricted to graphs with $\alpha(G) > |V|/2$.

To test the proposed approach, we generated a number of graphs with $\alpha(G) > |V|/2$ using Sanchis generator of maximum clique instances available from the DIMACS ftp server at ftp://dimacs.rutgers.edu/pub/challenge/ (see also [91]). Sanchis graph generator details may be found in [79, 123]. We took complements of graphs obtained using the generator, and tested them for connectivity. All graphs considered in our experiments were connected. Since the Sanchis generator produces graphs with a predetermined maximum clique size $\omega(G)$, the size $\alpha(\bar{G})$ of the maximum independent set of its complement $\bar{G}$ is known, $\alpha(\bar{G}) = \omega(G)$. The number of vertices in the graphs used in our computations ranged from 1000 to 18000. Due to their large size, the maximum independent set problem in these graphs cannot be solved using standard exact algorithms. However, the critical set approach presented in this paper was extremely effective with all of the considered cases. The results of computations are summarized in Table 1. Here each row corresponds to one graph $G = (V, E)$ and other notations used in the table are defined as follows. U_c denotes the computed critical set, I_c is the critical independent set obtained from U_c by taking the isolated vertices of $G(U_c)$, and, as before, $\alpha_c(G) = |I_c| - |N(I_c)|$. By $G_r = (V_r, E_r)$ we denote

the output of our algorithm, i.e., the graph obtained from G after recursively removing a critical independent set and its neighborhood. Finally, the last column reports the CPU time (in seconds) of execution of the proposed algorithm implemented in C programming language. The programs were compiled with the GCC 3.3.6 compiler and run on a Dell Inspiron 8600 computer running Linux 2.6 and configured with Pentium-4M 1400 MHz processor and 512 MB of RAM.

In another set of numerical tests we experimented with Erdös collaboration networks available from Batagelj's Networks/Pajek Graph Files website [20], as well as complements of the graph coloring problem instances from the Trick's graph coloring page [131]. The results of these experiments are presented in Tables 2 and 3, where all the notations used are the same as in Table 1. In Erdös collaboration networks considered, the vertices represent authors who are "connected" to Paul Erdös through a short path of co-authors [19]. More specifically, Table 2 considers instances with names in the form ERDOS.$x.y$, where x represents the last two digits of the year for which the network was constructed, and y represents the largest Erdös number of an author represented by a vertex in the graph. For example, the vertices in graph ERDOS.99.1 represent 472 co-authors of Paul Erdös, and the vertices in graph ERDOS.99.2 correspond to 6100 researchers who co-authored a paper either with Erdös or with at least one of his co-authors. We considered such networks for years 1997-1999 and $y = 1$ and 2. Note that in all considered instances of Erdös collaboration networks their independence number exceeds half of the number of vertices, which is typical for social networks, as well as for large sparse networks arising in many other applications [3, 80, 81]. Naturally, the approach proposed in this chapter is a very effective step in solving the maximum independent set problem for such networks. On the other hand, complements of most of the standard DIMACS maximum clique instances have relatively small independence numbers [91] and empty critical

Table 1 Results of experiments with Sanchis graphs

$\|V\|$	$\|E\|$	$\alpha(G)$	$\|U_c\|$	$\|I_c\|$	$\alpha_c(G)$	$\|V_r\|$	$\|E_r\|$	Time
1000	181256	524	524	524	476	0	0	0.07
1000	186723	505	505	505	495	0	0	0.04
2000	686341	1103	1103	1103	897	0	0	0.97
2000	711955	1067	1067	1067	933	0	0	0.44
3000	944175	1535	1535	1535	1465	0	0	0.58
3000	954717	1563	1563	1563	1437	0	0	0.87
4000	1014603	2069	2069	2069	1931	0	0	7.08
4000	1090563	2309	2309	2309	1691	0	0	11.27
5000	1533472	2717	2717	2717	2283	0	0	11.43
5000	720845	3132	3132	3132	1868	0	0	47.74
6000	1775988	3302	3305	3259	2741	46	44	86.97
6000	1815973	3412	3412	3412	2588	0	0	33.64
7000	890777	4493	4493	4493	2507	0	0	154.07
8000	3193335	4394	4394	4394	3606	0	0	103.30
8000	481800	5249	5249	5249	2751	0	0	263.44
9000	4040615	4927	4930	4887	4113	43	41	273.70
9000	681131	5899	5899	5899	3101	0	0	381.69
10000	3775385	5811	5813	5799	4201	14	12	625.58
10000	4908379	5507	5508	5478	4522	30	29	249.76
11000	2546883	6862	6862	6862	4138	0	0	594.24
11000	6528244	5901	5902	5868	5132	34	33	220.11
12000	4862197	7098	7097	7075	4925	22	20	1041.04
12000	5549355	6973	6973	6973	5027	0	0	463.30
13000	1339999	8474	8474	8474	4526	0	0	1131.41
13000	5638263	7698	7705	7640	5358	67	58	1865.82
14000	10772525	7417	7423	7346	6654	77	72	1045.77
14000	3371180	8844	8844	8844	5156	0	0	1330.66
15000	4207335	9413	9417	9386	5614	31	27	2288.17
15000	6912205	8993	8994	8983	6017	11	10	1523.01
16000	14346706	8401	8407	8360	7640	47	41	2923.90
16000	4807361	10042	10042	10042	5958	0	0	2215.34
17000	10748092	9898	9901	9862	7138	39	36	2666.40
17000	913028	11239	11239	11239	5761	0	0	2596.95
18000	2038675	11782	11782	11782	6218	0	0	3249.14
18000	5106081	11412	11412	11402	6594	14	10	3830.01

Table 2 Results of experiments with Erdös networks

Graph	$\|V\|$	$\|E\|$	$\alpha(G)$	$\|U_c\|$	$\|I_c\|$	$\alpha_c(G)$	$\|V_r\|$	$\|E_r\|$	Time
ERDOS.97.1	472	1314	254	317	179	58	158	271	11.96
ERDOS.97.2	5488	8972	5047	5034	5034	4606	26	13	183.52
ERDOS.98.1	485	1381	261	366	152	58	229	526	1.00
ERDOS.98.2	5822	9505	5368	5356	5356	4914	24	12	213.02
ERDOS.99.1	492	1417	263	375	144	56	246	614	0.42
ERDOS.99.2	6100	9939	5639	5629	5629	5178	20	10	239.61

independent sets, thus the critical independence set approach is useless for these instances. The same can be said about the collections of test instances for the maximum independent set problem available online at

http://www.research.att.com/~njas/doc/graphs.html and

http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm

For all of these instances, with exception of several instances that had isolated vertices, the computed critical independent sets were empty. The results reported in Table 3 show that for a critical independent set to be nonempty, the independence number does not necessarily need to be very large, as for some of the considered graphs $\alpha(G) < |V|/2$.

Table 3 Results of experiments with coloring problem instances

Graph	$\|V\|$	$\|E\|$	$\alpha(G)$	$\|U_c\|$	$\|I_c\|$	$\alpha_c(G)$	$\|V_r\|$	$\|E_r\|$	Time
anna	138	493	80	75	61	29	45	45	0.01
david	87	406	36	81	13	9	70	305	0.01
fpsol2.i.1	496	11654	307	496	227	227	269	11654	0.31
fpsol2.i.2	451	8691	261	348	172	120	223	977	0.33
fpsol2.i.3	425	8688	238	322	146	94	223	974	0.27
huck	74	301	27	34	16	4	46	143	0.01
inithx.i.1	864	18707	566	741	430	363	367	11079	2.21
inithx.i.2	645	13979	365	383	257	144	273	642	0.97
inithx.i.3	621	13969	360	349	259	132	221	423	0.86
jean	80	254	38	39	26	15	35	108	0.01
zeroin.i.1	211	4100	120	211	85	85	126	3775	0.02
zeroin.i.2	211	3541	127	143	111	59	48	207	0.04
zeroin.i.3	206	3540	123	140	108	56	46	206	0.04

CHAPTER IV

RELATIONSHIP BETWEEN THE CRITICAL SET METHOD AND OTHER SCALE-REDUCTION TECHNIQUES

This chapter compares several different approaches used to reduce the size of an instance of the maximum independent set problem that were proposed in the literature since 1975. Section IV.1 introduces the problem and general characteristic of considered reductions. Next, Sections IV.2-IV.5 present the scale-reduction approaches of interest. Then, Section IV.6 investigates the relationship between these approaches and shows that they are equivalent in some sense. Finally, Section IV.7 highlights the differences between the considered methods and discusses their possible extensions.

IV.1. Scale Reduction in the Maximum Weight Independent Set Problem

Consider the decision version of the maximum weight independent set problem: given a simple undirected vertex-weighted graph $G = (V, E, w)$ and a parameter k, decide if G contain an independent set of weight at least k. All the approaches compared in this chapter may be viewed as scale-reduction techniques based on the following idea: given a problem instance G and a parameter k, we can reduce (G, k) to another instance of the same problem (G', k') such that (G, k) is a "yes" instance if and only if (G', k') is a "yes" instance and $|V(G')| < |V(G)|$. Simply speaking, the problem of finding a maximum weight independent set is reduced to the same problem on a graph with a smaller number of vertices. This reduction technique is also referred to as *kernelization* in [43,46], where the irreducible part G' is called the instance's *kernel*, which is what makes the problem hard to solve. The following section will present different approaches for the MWISP size reduction. In the next section the relation

between presented approaches will be stated and their equivalence will be defined. Finally, the differences between approaches and possible extensions will be considered at Section IV.7 and one of such possible extensions, based on t-hat structure, will be presented at Section IV.8.

IV.2. IP Relaxation Based Approach

The first approach of interest was presented by Nemhauser and Trotter [110] in 1975. The MWISP was formulated as the following integer program, also known as the *edge formulation*:

$$\begin{aligned} \alpha_w(G) = \max \quad & \sum_{i \in V} w_i x_i, \\ \text{s.t.} \quad & x_i + x_j \leq 1 \quad \forall (i,j) \in E, \\ & x_i \in \{0,1\} \quad \forall i \in V. \end{aligned} \tag{4.1}$$

Its optimal solution $x_I \in \{0,1\}^{|V|}$ is the characteristic vector of a maximum weight independent set I of G, i.e., such that $v \in I$ if and only if $x_v = 1$. The linear programming relaxation of the above integer program is given by

$$\begin{aligned} \max \quad & \sum_{i \in V} w_i x_i, \\ \text{s.t.} \quad & x_i + x_j \leq 1 \quad \forall (i,j) \in E, \\ & 0 \leq x_i \leq 1 \quad \forall i \in V. \end{aligned} \tag{4.2}$$

Nemhauser and Trotter proved that any optimal solution x^* of (4.2) is such that $x_i^* \in \{0, 1, 1/2\},\ i = 1, \cdots, n$. They also considered the partition of graph vertices $(V_0, V_1, V_{1/2})$ corresponding to x^*, where $i \in V_j$ if $x_i^* = j, i = 1, \cdots, n, j \in \{0, 1/2, 1\}$, and proved that there exists a maximum weight independent set that contains all vertices from V_1. Thus, the original problem could be reduced to the MWISP on

$G' = G[V_{1/2}] = G - V_1 - V_0$. The authors also provided an efficient algorithm to find the sets V_0, V_1 and $V_{1/2}$. This algorithm will be referred to as $\mathcal{A}_{NT}$ and the notation $G' = \mathcal{A}(G)$ will mean that G' is the reduced graph obtained by applying algorithm $\mathcal{A}$ to the graph G.

IV.3. Approach Based on Roof Duality

Hammer et al. [32, 33, 77] used pseudo-boolean optimization for solving discrete optimization problems, including the MWISP. For $\mathbb{B} = \{0, 1\}$, the function $f : \mathbb{B}^n \to \mathbb{R}$ is called *pseudo-boolean*. In this subsection, we will first explain a more general scale-reduction technique for pseudo-boolean optimization based on the concept of roof duality, and then discuss the application of this technique to the MWISP.

IV.3.1. Roof Duality Essentials

For a problem

$$\max_{x \in \mathbb{B}^n} f(x), \tag{4.3}$$

replacing $f(x)$ by an *upper plane* $p(x)$, i.e., a linear function such that $p(x) \geq f(x)\ \forall x \in \mathbb{B}^n$, yields a linear relaxation of the problem (4.3) given by

$$\max_{x \in \mathbb{B}^n} p(x). \tag{4.4}$$

Let S denote a set of upper planes of $f(x)$. S is called *complete* if $f(x) = \min_{p(x) \in S} p(x)$. If S is complete, then the optimal objective value of problem (4.3) is equal to $\max_{x \in \mathbb{B}^n} \min_{p(x) \in S} p(x)$. Since this new problem is as hard as the original one, it is of interest to consider the problem

$$\min_{p(x) \in S} \max_{x \in \mathbb{B}^n} p(x), \tag{4.5}$$

which is called the *plane dual*. Note that the optimal objective value of (4.5) is no less than that of (4.3).

Consider the problem (4.3) with a quadratic objective:

$$\max_{x\in\mathbb{B}^n} f(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij}x_i x_j, \tag{4.6}$$

where $q_{ij} = 0$ whenever $i > j$, and since $q_{ii}x_i x_i = q_{ii}x_i$, the linear terms are not considered separately. In this case, the upper plane $p(x)$ can be generated as sum of *local* upper planes $p_{ij}(x_i, x_j) = a_{ij}x_i + b_{ij}x_j + c_{ij}$ built for each term $q_{ij}x_i x_j$. The closest to $q_{ij}x_i x_j$ local upper plane is called a *tile*. It is evident that more than one tile may be generated for each term. Moreover, there are infinitively many tiles that may be generated for each term, but it is possible to explicitly write down the analytical expression for tile using parameter $\lambda_{ij} \in [0, q_{ij}]$ [77]. The upper plane built of tiles has the form $p(x, \lambda) = \sum_{i,j=1}^{n} p_{ij}(x_i, x_j, \lambda_{ij})$ and is called a *roof*. The problem (4.5) constructed using roof as upper planes is called the *roof dual* problem. The roof-dual bound can be determined efficiently by maximum flow computations in the implication network corresponding to the problem [77]. Some variables of an optimal solution to the roof dual problem may have integer values (binary in this case). For some index $j \in V$ and binary value $\alpha \in \mathbb{B}$, *strong* (respectively, *weak*) *persistency* holds for problem $\min\{f(x) : x \in \mathbb{B}^n\}$ if $x_j = \alpha$ holds for all (respectively, for some) optimal solutions of this problem.

IV.3.2. Roof Duality and the Maximum Weight Independent Set Problem

In [33], the maximum independent set problem is formulated as the following quadratic unconstrained pseudo-boolean problem:

$$\alpha(G) = \max_{x \in \mathbb{B}^n} \sum_{v \in V} x_v - \sum_{(i,j) \in E} x_i x_j. \tag{4.7}$$

Its weighted extension can be written as

$$\alpha_w(G) = \max_{x \in \mathbb{B}^n} \sum_{v \in V} w_v x_v - M \sum_{(i,j) \in E} x_i x_j, \tag{4.8}$$

where M is a sufficiently large penalty coefficient (e.g., $M = w(V)$). The authors of [33,77] derived the persistency criteria for the solutions of the roof dual problem. After applying roof duality method to the above quadratic formulation of the maximum weight independent set problem (4.7), the original graph may be partitioned into three disjoint subsets $(V_0, V_1, V \setminus V_0 \setminus V_1)$, where V_0 and V_1 correspond to the persistent assignments of $x_i = 0$ and $x_i = 1$, respectively. So, the problem can be reduced by eliminating vertices from V_0 and V_1, and the corresponding algorithm will be denoted as $\mathcal{A}_{RD}$.

IV.4. Crown Structure Elimination

Chlebík and Chlebíkova [46] introduced the concept of crown structure for vertex-weighted graphs as follows. Given a graph $G = (V, E, w)$, a *crown* (respectively, a *strong crown*) in G is a subgraph C induced by $I \cup S$ in G, where I is an independent set, S is the set of neighbors of I in G, and $w(N(U) \cap I) \geq w(U)$ (respectively, $w(N(U) \cap I) > w(U)$) holds for every nonempty set $U \subseteq S$. If graph G is unweighted, then the following equivalent definition by Chen and Zhang [43] provides intuitive understanding of the crown structure: the *crown structure* is a subgraph C induced by $I \cup S$ in G where I is an independent set, S is the set of neighbors of I in G and $|I| \geq |S|$ and there is a matching between I and S that saturates S, i.e., all vertices in S are in the matching. If $|I| > |S|$, the crown is called a *proper crown* or *strong*

crown. Figure 5 shows a graph that is a crown.

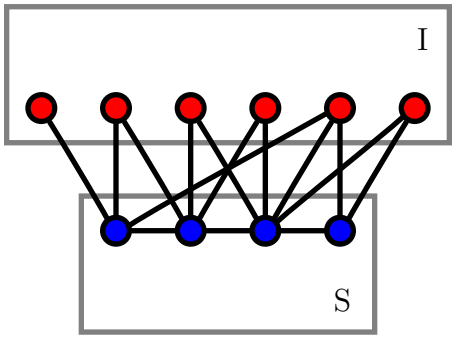


Fig. 5 Crown structure

The scale reduction in this case is done by eliminating proper crown structures from the original graph. The paper [46] demonstrates that the part I in the definition of the crown structure is a part of a maximum weight independent set, so no vertices from part S can belong to such a maximum independent set. The corresponding partition is given by $(I, S, V \setminus I \setminus S)$ and is called a *crown decomposition*. Again, the algorithm is applied in iterative fashion until all proper crowns are eliminated from the graph. The remaining graph is called *crown-free* and has only the trivial crown decomposition $(\emptyset, \emptyset, V)$. The algorithm utilizing the crown elimination decomposition will be denoted by $\mathcal{A}_{CE}$.

IV.5. Approach Based on Critical Sets

To be consistent, we restate and fit to the current format our result that allows to utilize critical weighted independent set in order to reduce the size of the MWISP. After a critical weighted independent set I_c is found in G, the similar vertex partition may be introduced: $(I_c, N(I_c), V \setminus I_c \setminus N(I_c))$. According to our result, there exists a maximum weight independent set I such that the vertices from the first partition

I_c are a part of I (i.e., $x_i : i \in I_c$ are fixed to 1), therefore the vertices from the second partition are not in the set I, and the original graph could be reduced to $G' = G - I_c - N(I_c)$.

Denote by $\mathcal{A}_{CS}$ the iterative algorithm based on the critical weighted independent set elimination, i.e., the algorithm applied to the reduced graph until the critical weighted independent set is empty.

IV.6. Relation Between Approaches

For the set of algorithms $\{\mathcal{A}_i : i \in \{NT, RD, CE, CS\}\}$, let us define the following equivalence relation:

$$\mathcal{A}_i \sim \mathcal{A}_j \Leftrightarrow \mathcal{A}_j(\mathcal{A}_i(G)) = \mathcal{A}_i(G) \;\&\; \mathcal{A}_i(\mathcal{A}_j(G)) = \mathcal{A}_j(G)$$

Simply speaking, the algorithms $\mathcal{A}_i$ and $\mathcal{A}_j$ are equivalent if no additional reduction may be obtained by applying algorithm $\mathcal{A}_j$ after $\mathcal{A}_i$ and vice versa. The defined relation is reflective ($\mathcal{A}_i \sim \mathcal{A}_i$), symmetric ($\mathcal{A}_i \sim \mathcal{A}_j \Leftrightarrow \mathcal{A}_j \sim \mathcal{A}_i$) and transitive ($\mathcal{A}_i \sim \mathcal{A}_j \;\&\; \mathcal{A}_j \sim \mathcal{A}_k \Rightarrow \mathcal{A}_i \sim \mathcal{A}_k$), so this is an equivalence relation. The relation between the four approaches described above is given by the next proposition.

Proposition 7. *Algorithms presented by Nemhauser and Trotter ($\mathcal{A}_{NT}$), Boros, Hammer and Tavares ($\mathcal{A}_{RD}$), Chen and Zhang ($\mathcal{A}_{CE}$), and Butenko and Trukhanov ($\mathcal{A}_{CS}$), are equivalent with respect to the equivalence relation defined above.*

Proof.

- $\mathcal{A}_{NT} \sim \mathcal{A}_{CE}$

  Proved by Chen and Zhang [43] for unweighted case and by Chlebík and Chlebíková [46] for the general case.

- $\mathcal{A}_{CS} \sim \mathcal{A}_{CE}$

  The following observations are valid:

– A non-empty critical weighted independent set with a positive critical number is a proper crown. Assume $I_c \subseteq V$ is a critical weighted independent set, but is not a proper crown. Then there exists $U \subseteq N(I_c)$ such that $w(N(U) \cap I_c) > w(U)$. Let $J = N(U) \cap I_c$ and consider set $I_c \setminus J$ which is also an independent set. We have:

$$\begin{aligned} w(I_c \setminus J) - w(N(I_c \setminus J)) &= w(I_c) - w(J) - w(N(I_c \setminus J)) \\ &= w(I_c) - w(J) - w(N(I_c)) + w(N(J) \cap N(I)), \end{aligned}$$

since $N(I_c \setminus J) \subseteq N(I_c)$. Also, since $N(J) = N(N(U) \cap I_c)$, $U \subseteq N(I_c)$ and $I_c \setminus J \subseteq I_c$, we have

$$U \subseteq N(J) \subseteq N(I_c)$$

and

$$w(U) \leq w(N(J)) \leq w(N(I_c)).$$

Thus,

$$\begin{aligned} & w(I_c \setminus J) - w(N(I_c \setminus J)) \\ & \geq w(I_c) - w(J) - w(N(I_c)) + w(U \cap N(I_c)) \\ & = w(I_c) - w(N(I_c)) - w(J) + w(U) > w(I_c) - w(N(i_c)), \end{aligned}$$

which contradicts to the assumption that I_c is a critical weight set.

– A crown (even a proper one) is not necessarily a critical weight independent set, since there is no maximality requirement on the difference between weight of the set of vertices and its neighborhood.

– If the graph contains a proper crown, then this graph contains a non-empty

critical weighted independent set with positive critical number, since

$$\begin{aligned}&\max\{w(I) - w(N(I)) \ : \ I \text{ is an independent set in } G\}\\ &\geq \max\{w(C) - w(N(C)) \ : \ C \text{ is a proper crown in } G\},\end{aligned}$$

which is greater than zero.

- $\mathcal{A}_{CS}(\mathcal{A}_{CE}(G)) = \mathcal{A}_{CE}(G)$. If we assume the opposite, then $\mathcal{A}_{CS}$ has found a non-empty critical weighted independent set I_c in $\mathcal{A}_{CE}(G)$, but I_c is also a proper crown, which leads to a contradiction, since $\mathcal{A}_{CE}$ has eliminated all proper crowns from G.
- $\mathcal{A}_{CE}(\mathcal{A}_{CS}(G)) = \mathcal{A}_{CS}(G)$. If we assume the opposite, then $\mathcal{A}_{CE}$ has found a proper crown C in $\mathcal{A}_{CS}(G)$. But this implies that there is a non-empty critical weighted independent set in $\mathcal{A}_{CS}(G)$ with positive critical number. This contradicts to the fact that $\mathcal{A}_{CS}$ has eliminated all such sets.

So, the proper crown elimination procedure and the critical weighted independent set procedure produce the same result: the new graph is a proper crown free graph, and $\mathcal{A}_{CE} \sim \mathcal{A}_{CS}$. It should be noted that it is not guarantied that the new graph will be the same in both cases (even when applying critical weighted independent set detection algorithm to the same graph more than once starting from different vertices, it is possible to obtain different resulting sets).

- $\mathcal{A}_{CS} \sim \mathcal{A}_{RD}$

  Consider the graph $B'(G)$ that is used in the critical set algorithm. $B'(G)$ is constructed from $B(G)$ by adding two vertices s and s' (source and sink) and a set of edges $\{(s, v), (v', s') : \forall v \in V, v' \in V'\}$. The original edges of $B(G)$ will be directed from V to V'. Edge weight will be assigned as follow: $w_{sv} = w_{vs'} = w_v$,

and all other edges have weight $+\infty$ (or big M). Since $B'(G)$ is symmetric, it may be considered as an implication network [33] for some quadratic posiform. Using the procedure inverse to the implication network construction, the quadratic posiform corresponding to implication network $B'(G)$ is given by

$$\phi(x) = 2 \sum_{v \in V(G)} w_i x_0 \bar{x}_i + 2M \sum_{(i,j) \in E(G)} x_i x_j.$$

After excluding artificial variable x_0, we obtain:

$$\phi(x) = 2 \sum_{v \in V(G)} w_i (1 - x_i) + 2M \sum_{(i,j) \in E(G)} x_i x_j$$

According to [77] the maximum flow in implication network gives a fractional optimal solution for problem of minimization of the corresponding quadratic posiform over binary hypercube, i.e.

$$\begin{aligned} &\min_{x_i \in \{0,1\}} \phi(x) \\ &\qquad = \min_{x_i \in \{0,1\}} 2 \sum_{v \in V(G)} w_i (1 - x_i) + 2M \sum_{(i,j) \in E(G)} x_i x_j \\ &\qquad = 2w(V) + \min_{x_i \in \{0,1\}} (- \sum_{v \in V(G)} (-w_i x_i) + 2M \sum_{(i,j) \in E(G)} x_i x_j) \\ &\qquad \sim \max_{x_i \in \{0,1\}} \sum_{v \in V(G)} w_i x_i - M \sum_{(i,j) \in E(G)} x_i x_j, \end{aligned} \tag{4.9}$$

which is a quadratic binary formulation of the maximum weight independent set problem. The procedure of finding a weakly persistent assignment based on the implication network proposed in [33, 77] picks exactly the same vertices as those used for a critical weighted independent set construction in [39]. So, the partition obtained by $\mathcal{A}_{RD}$ is exactly the same as for $\mathcal{A}_{CS}$.

Note that from [77] it follows that $\mathcal{A}_{NT} \sim \mathcal{A}_{RD}$, so $\mathcal{A}_{CS} \sim \mathcal{A}_{RD}$ can be alternatively shown by transitivity.

- Finally, $\mathcal{A}_{NT} \sim \mathcal{A}_{RD} \sim \mathcal{A}_{CE} \sim \mathcal{A}_{CS}$ by transitivity.

□

IV.7. Extensions and Differences

Motivated by our work described in the previous chapter, Larsen [95] proposed a modified algorithm that allows to find a maximum weight critical independent set. The advantages of the approach include the following:

- more vertices may be eliminated (actually, not only proper, but also non-proper crowns crown will be eliminated);
- a maximum weight critical independent set needs to be found only once, and after this set is eliminated, the empty set is the only critical weighted independent set of the remaining graph (recall that the original algorithm $\mathcal{A}_{CS}$ eliminates critical independent sets recursively);
- the modified algorithm was used to develop a necessary and sufficient condition for existence of a non-empty critical weighted independent set in a given graph.

The disadvantage is the complexity of the modified algorithm, which is $|V|$ times higher than the complexity of the original algorithm.

Chen and Zhang [43] also extended their algorithm (unweighted case only) to eliminate non-proper crowns (crowns that have $|I| = |S|$). Part I belongs to some maximum independent set and part S is its neighborhood in non-proper crown case too. The result of this algorithm is the same as Larsen's algorithm result [95], with the same advantages, disadvantages and complexity. In addition, Chen and Zhang [43] consider a structure called t-hat, which is similar to crown, but $|I| - |S| = -t$ is negative. The main result shown is that for *minimal* t-hat, i.e. t-hat that has the property $|N(C)| - |C| \geq t \ \forall C \subset I$, the minimum vertex cover (maximum independent set)

problem is branchable on I (i.e., there exists a solution for the problem that contains I completely or not at all). The paper [43] has considered only the unweighted version of the problem, while all other aforementioned papers dealt with the weighted case.

Finally, the roof-duality approach is much more general and may be useful for many other quadratic unconstrained problem in addition to the maximum weight independent set problem.

IV.8. t-Hat Structures and Maximum Weight Independent Set

Recall that the authors of [43] defined the *t-hat structure* (C, H) in graph $G = (V, E)$ as a subgraph with vertex set $C \cup H$ in which C is an independent set, $H = N(C)$ is its neigborhood, and $|H| - |C| = t$. In such a graph, C is called the *cap* and H is called the *head* (Figure 6). A t-hat (C, H) is called *minimal* if for any $C' \subseteq C$, $|N(C')| - |C'| \geq t$. For $t \leq 0$ the t-hat is a crown, and for $t < 0$ it is a proper crown, so the crown elimination procedure may be applied to make the graph decomposition.

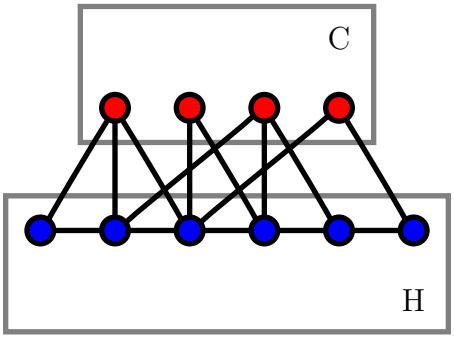


Fig. 6 t-hat structure

For $t > 0$ it is not possible to claim that part C belongs to a maximum independent set, but the authors provided the results that the maximum independent set problem is *branchable* on C, i.e., there exists a maximum independent set that either

contains all vertices from C or not at all.

The definition and result may be extended to the weighted graphs. First of all, the t-hat structure (C, H) in a weighted graph $G = (V, E, w)$ is a subgraph with vertex set $C \cup H$, where C is an independent set, $H = N(C)$, and $w(H) - w(C) = t$. In such definition t is not required to be integer. In the same way as in the original paper, the t-hat (C, H) is *minimal*, if $\forall C' \subseteq C$, $w(N(C')) - w(C') \geq t$. If $t \leq 0$, the results from [43] still hold for this case, i.e., in such case t-hat is a crown and could be eliminated from the graph when solving the MWISP, as shown in [46]. For the case when $t > 0$, the following theorem provides a result that extends the original one:

Theorem 8. *For $t > 0$, given a graph $G = (V, E, w)$ and a minimal t-hat (C, H) of G, the MWISP is branchable on C.*

Proof. The proof mostly follows the idea of the proof of the original theorem, extending the arguments to weighted case when needed.

Let some $C' \subset C$ be a part of I, a maximum weight independent set in G. Then $N(C') \subseteq N(I)$, and since (C, H) is minimal t-hat and $C' \subseteq C$, we have

$$\begin{aligned} w(H) - w(N(C')) &= t + w(C) - w(N(C')) \\ &\leq t + w(C) - (t + w(C')) \\ &= w(C) - w(C') \\ &= w(C \setminus C'). \end{aligned}$$

Now, let $J = (I \setminus H) \cup (C \setminus C')$. This set has the following properties:

- J is an independent set;
- $C' \subseteq J$ and $C \setminus C' \subseteq J$, thus, $C \subseteq J$;
- since $N(C') \subseteq N(I)$, $w(J) = w(I) - (w(H) - w(N(C'))) + w(C \setminus C') \geq w(I)$.

So, J is a maximum weight independent set that contains all vertices from C. □

The proved result allows one to speed up a branch-and-bound algorithm for the MWISP as follows: when the minimal t-hat structure (C, H) is detected in the graph, it is possible to make only two branches, putting all vertices from C to the maximum weight independent set together and then put all of them out, instead of doing branching on each vertex separatelly and creating $2^{|C|}$ branches.

CHAPTER V

CLIQUE RELAXATION MODELS*

This chapter concentrates on the clique relaxation models, corresponding optimization problems and their properties. Section V.1 discusses possible ways of relaxing the clique requirements and introduces models that are obtained as a result of such relaxations. Section V.2 establishes the complexity results for optimization problems related to the introduced models. In Section V.3, we provide mathematical programming formulations of the considered problems. Section V.4 emphasizes one particular case of the problem, which is easier to solve than the general case. Finally, Section V.5 presents the result of numerical experiments.

V.1. Motivation and Models

As was already mentioned, all vertices in a set that forms a clique have the maximum possible degree (the clique size minus one), the smallest possible diameter (one), and the smallest possible distance between any pair of vertices (one). By relaxing one of the properties (degree, distance, or diameter), clique relaxation models arise.

Relaxing the restriction on the distance between vertices yields the *k-clique* model:

For a graph G, a k-clique is a subset of vertices C such that $\forall i, j \in C,\ d_G(i, j) \leq k$.

By relaxing the requirement on the diameter of the induced subgraph, the *k-club*

*Parts of this chapter are reprinted with permission from Balasundaram, B., Butenko, S., Trukhanov, S.: Novel approaches for analyzing biological networks. Journal of Combinatorial Optimization **10**(1), 23–39 (2005) © Springer.

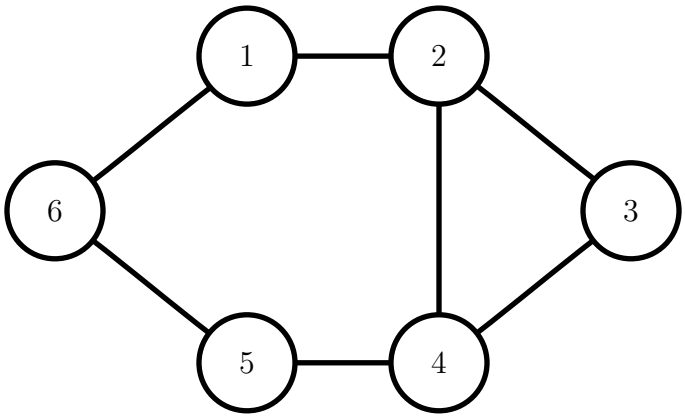


Fig. 7 2-club and 2-clique example [7]

definition is obtained.

For a graph G, a vertex subset D is called a k-club if $diam(G[D]) \leq k$.

It is clear that a k-club is also a k-clique, but the opposite is not necessarily true, since the shortest distance between two vertices is the original graph (as required for k-clique) does not guarantee the same distance between these vertices in the induced subgraph as required for a k-club.

To highlight the differences between these structures, we turn to the graph in Figure 7, that first appeared in Alba [7] and was subsequently adopted by other authors. In this graph, the maximal 2-cliques are given by $C_1 = \{1, 2, 3, 4, 5\}$ and $C_2 = \{1, 2, 4, 5, 6\}$. It is easy to see that C_1 is not a 2-club, since the diameter of induced subgraph $G(C_1)$ is 3. The 2-clubs of this graph are $D_1 = \{1, 2, 3, 4\}$, $D_2 = \{2, 3, 4, 5\}$ and $D_3 = C_2$. A study of relations between cliques, clans and clubs in a graph can be found in [107].

Finally, a degree-based relaxation called *k-plex* is defined as follows:

A subset of vertices S is called a k-plex if for each vertex $v \in S$, $deg_{G[S]}(v) \geq |S| - k$.

In other words, each vertex of k-plex may have at most k non-neighbors from

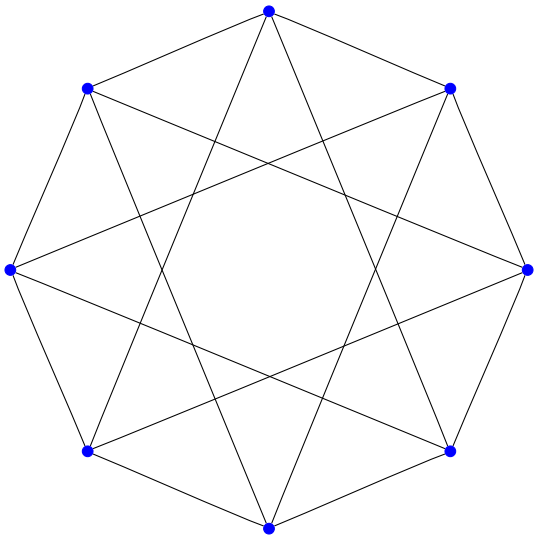


Fig. 8 Triangle free 4-plex

this k-plex. Figure 8 presents a 4-plex consisting of 8 vertices.

Similar to the relation between the cliques and independent sets, the complement of a k-plex is called a *co-k-plex*, but at the same time there is no meaningful complementary for the k-clique and k-club models. Formally, a set of vertices C is called a co-k-plex, if for any $i \in C$, $deg_{G[C]}(i) < k$.

The corresponding optimization problems that require to find a maximum weight k-clique (k-club, k-plex) are formulated as follows: given a graph G with vertex weights given by a vector w, find a k-clique (k-club, k-plex) of maximum weight in G. The optimal solution values of these problems are denoted by $\tilde{\omega}_k(G)$, $\bar{\omega}_k(G)$, $\omega_k(G)$ and are called the k-clique, k-club and k-plex number of graph G, respectively. The maximum co-k-plex problem is defined in the same fashion, and its optimal solution value is denoted by $\alpha_k(G)$. An obvious observation is that with increasing k, the k-clique (k-club, k-plex) number is increasing too, since a k-clique (k-club, k-plex) is also an m-clique (m-club, m-plex), for all $m > k$.

The clique relaxation models were first introduced in social network analysis as

a replacement for a clique, since the social networks are often built based on inexact data and usually have a small maximum clique, that may not be useful in analysis. Hence, since the clique relaxations overcome this issue, a logical question is, how much larger can the weight of a k-clique (k-club, k-plex) be compared to that of the maximum weight clique? For a k-club (and, thus, for a k-clique) the answer is straightforward: it is possible to find graphs with any given maximum clique size that have arbitrarily large k-club. The star $K_{1,n}$ is one such example, since it is a k-club for any $k \geq 2$, but its maximum clique size is only 2.

As for the k-plex number, we establish the following theorem to answer our question.

Theorem 9. *For any graph G and a positive integer k,*

$$\alpha_k(G) \leq k\alpha(G) \tag{5.1}$$

and

$$\omega_k(G) \leq k\omega(G). \tag{5.2}$$

Proof. Assume, that the theorem statement is incorrect, then there exists a graph G with $\alpha(G) = m$ and $\alpha_k(G) > km$ for some positive integer m. Let G' be a maximum co-k-plex in G. Apply Algorithm 2, which is a simple greedy approach, to find an independent set I in G'. At each step, this algorithm maintains an independent set I and the set N, that is the neighborhood of I. At each iteration, the algorithm picks a vertex u with the maximum weight among the remaining vertices and adds it to I (line 5 of Algorithm 2), so

$$w(u) \geq w(v) \ \forall v \in N_{G'-N-I}(u),$$

Algorithm 2 Greedy algorithm for the Maximum Weight Independent Set problem

```
1: procedure GreedyMWIS(G)
2:     I ← ∅
3:     N ← ∅
4:     while V(G) \ N \ I ≠ ∅ do
5:         u ← arg max{w(u) : u ∈ V(G) \ N \ I}
6:         I ← I ∪ {u}
7:         N ← N ∪ N(u)
8:     end while
9:     return I
10: end procedure
```

and

$$|N_{G'-N-I}(u)| \leq |N_G(u)| \leq k-1,$$

since G' and all its induced subgraphs are co-k-plexes. Thus,

$$(k-1)w(u) \geq w(N_{G'-N-I}(u)),$$

which implies that $(k-1)w(I) \geq w(N)$ at any step of the algorithm. The algorithm terminates when $V(G') \setminus N \setminus I$ is empty, which is equivalent to $I \cup N = V(G')$. Moreover, since $I \cap N = \emptyset$, we obtain:

$$(k-1)w(I) \geq w(N) \Rightarrow kw(I) \geq w(N) + w(I) = w(V(G')) > km.$$

So, $\alpha(G) \geq \alpha(G') \geq w(I) > m$, that contradicts to the initial assumption.

Finally,

$$\omega_k(G) = \alpha_k(\bar{G}) \leq k\alpha(\bar{G}) = k\omega(G).$$

□

Note that the bounds obtained in Theorem 9 are sharp for any k. As an example, the unweighted graph that is a disjoint union of m cliques of order k is a co-k-plex of order km, but only one vertex from each clique may be included in any independent set, so the stability number of this graph is just m. For the graph in Figure 8 and

$k = 4$ the bound is also exact, since its 4-plex number is 8 and its clique number is 2, since the graph is triangle free.

When defining clique relaxations, only one of three original properties is required to be preserved in the relaxed structure, so it is of interest to see what happens with the other two properties in the clique relaxation models. As it was shown before, a k-clique guarantees neither a small diameter nor a high vertex degree. The k-club, being also a k-clique, preserves the distance between vertices, but does not guarantee a high vertex degree ($K_{1,n}$, the star on $n + 1$ vertices, is a good example). For the k-plex, Seidman and Foster [125] established the fact that for $n \geq 2k - 1$ the k-plex of order n has the diameter at most 2. So, a large enough k-plex has a small diameter and is also a 2-club and a 2-clique.

V.2. Complexity Results

Before stating the complexity results, we introduce the recognition version of each problem. The k-Clique (k-Club, k-Plex) problem is defined as follows: Given a graph $G = (V, E)$ and positive integers k and m, does there exist an k-clique (k-club, k-plex) of size $\geq m$ in G? For a weighted graph, the corresponding problems are stated in terms of the weight instead of the size.

Theorem 10. *The k-Clique and k-Club problems are NP-complete for any fixed positive integer k.*

Proof. To prove NP-completeness of a problem $\mathcal{P}$, it suffices to show that [66]

1. $\mathcal{P} \in NP$;
2. Some known NP-complete problem is polynomially reducible to $\mathcal{P}$.

Note that for $k = 1$ both problems coincide with Clique problem, which is a

well-known NP-complete problem. So, we consider $k > 1$. Given a "yes" instance of k-CLIQUE (k-CLUB), any k-clique (k-club) of size $\geq m$ can be used as a certificate to verify that this is indeed a "yes" instance in polynomial time. Thus, k-CLIQUE and n-CLUB are in NP. To complete the proof, we reduce CLIQUE, which is a well known NP-complete problem, to k-CLIQUE (k-CLUB). Let $G = (V, E)$ be an instance of CLIQUE, which, we assume, does not contain isolated vertices, as no isolated vertex can be included in any clique of size two or more. We construct a corresponding instance of k-CLIQUE (k-CLUB) which is a $(\lfloor k/2 \rfloor + 2)$-partite graph $G' = (V', E')$. We define the vertex set as a union of $\lfloor k/2 \rfloor$ copies of V, a copy of E and one more auxiliary vertex 0:

$$V' = \bigcup_{i=1}^{\lfloor k/2 \rfloor} V^{(i)} \cup E \cup \{0\},$$

where $V^{(i)} = \{1^{(i)}, 2^{(i)}, \ldots, n^{(i)}\}$ is the i-th copy of V, $i = 1, \ldots, \lfloor k/2 \rfloor$. For any $v \in V$, by $v^{(i)} \in V^{(i)}$ we denote the i-th copy of v. The edge set connects copies of the same vertex in $V^{(i)}$ and $V^{(i+1)}$, $i = 1, \ldots, \lfloor k/2 \rfloor - 1$. A vertex $v^{\lfloor k/2 \rfloor}$ in $V^{\lfloor k/2 \rfloor}$ is connected to a vertex $e \in E$ if v is an endpoint of e in G. Finally, all vertices from E in G' are connected to 0. To summarize,

$$\begin{aligned} E' = & \bigcup_{i=1}^{\lfloor k/2 \rfloor - 1} \{(v^{(i)}, v^{(i+1)}) : v \in V\} \\ & \cup \{(v^{(\lfloor k/2 \rfloor)}, e) : v \in V, v \text{ is an endpoint of } e \text{ in } G\} \\ & \cup \{(e, 0) : e \in E\}. \end{aligned}$$

Figure 9 shows graph G' corresponding to graph G from Figure 7 for $k = 5$. Graph G' contains $\lfloor k/2 \rfloor |V| + |E| + 1$ vertices and can obviously be constructed in time polynomial with respect to the size of G.

Our reduction is based on the observation that G has a clique of size m if and only if G' has an k-clique (k-club) of size $m + (\lfloor k/2 \rfloor - 1)|V| + |E| + 1$. Note that G'

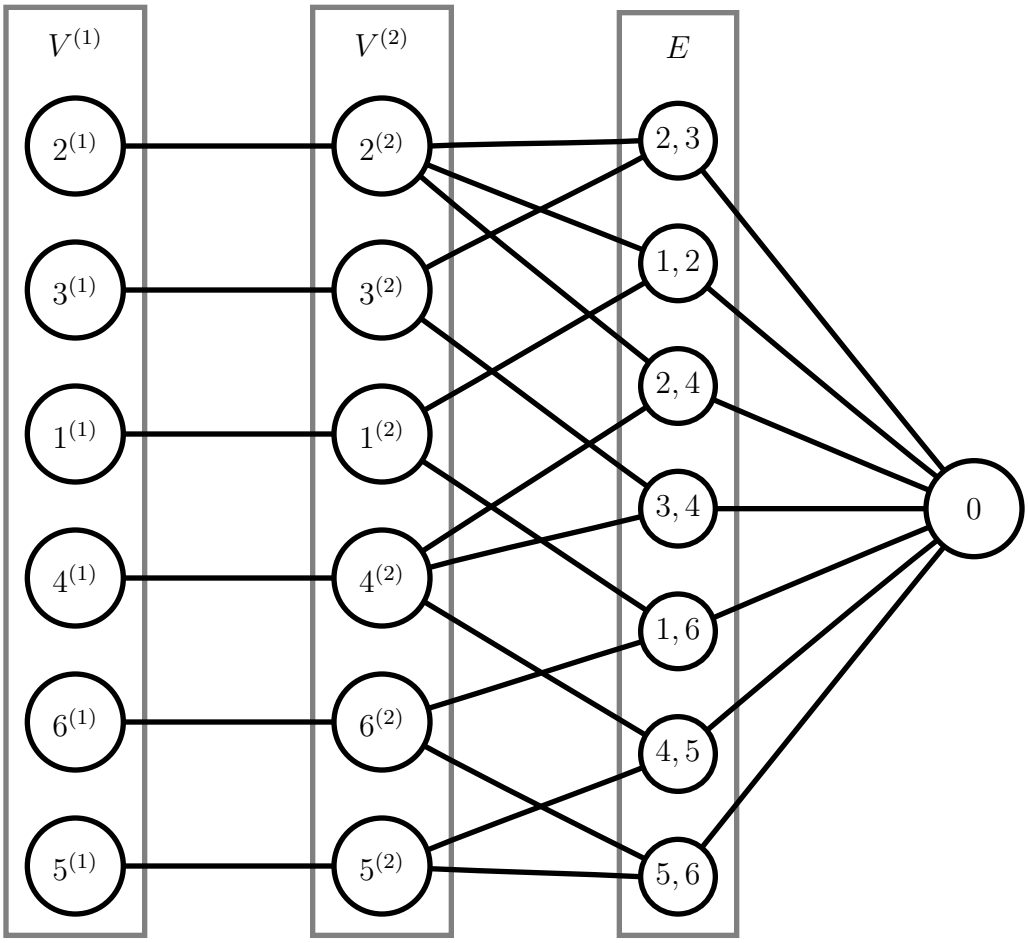


Fig. 9 An illustration to the proof of NP-completeness of the k-Club problem, for $k = 5$

is connected and $diam(G) \leq 2(\lfloor k/2 \rfloor) + 2$. Indeed, in G', all vertices of $V' \setminus V^{(1)}$ can be included in any k-clique (k-club). Two vertices $u^{(1)}, v^{(1)} \in V^{(1)}$ belong to the same k-clique (k-club) in G' if and only if $(u, v) \in E$ in G. Thus, k-Clique and k-Club are NP-complete problems for any positive integer k. □

Theorem 11. *The k-Plex problem is NP-complete for any fixed positive integer k.*

Proof. Proof can be found in [14, 15], or, alternatively, since the property of a graph "to be a k-plex" is non-trivial, interesting and holds on the induced subgraphs, the k-Plex problem complexity directly follows from the result by Yannakakis [140]. □

It is known that many massive networks arising in various applications have a relatively small diameter. This observation is commonly referred to as the *small world phenomenon* [105, 136]. Therefore, the clustering problems on graphs of small diameter are of particular interest. This motivates us to consider the k-Clique and k-Club problems on graphs of fixed diameter. Note that if $diam(G) \leq k$ then both the maximum k-clique problem and the maximum k-club problem are trivial as G is the maximum k-clique (k-club), therefore we are only interested in the case where $diam(G) > k$. For any $d > k$, we define the k-Clique(d) (k-Club(d)) problem as follows: Given a graph G of diameter d and positive integers k and m, does there exist an k-clique (k-club) of size $\geq m$ in G?

Theorem 12. *For any fixed positive integer n and $d > k$, the k-Clique(d) and k-Club(d) problems are NP-complete.*

Proof. Obviously both considered problems are in NP. To complete the proof we reduce Clique to k-Clique(d) and k-Club(d). We first prove the statement for $k = 1$. Given $G = (V, E)$ with no isolated vertices and $d > 1$, we construct a graph

$\hat{G} = (\hat{V}, \hat{E})$ of diameter d as follows.

$$\begin{aligned} \hat{V} &= V \cup \{u_i : i = 1, \ldots, d\}; \\ \hat{E} &= E \cup \{(v, u_1) : v \in V\} \cup \{(u_i, u_{i+1}) : i = 1, \ldots, d-1\}. \end{aligned}$$

Then G has a clique of size m if and only if $\hat{G}$ has a clique of size $m+1$ and the proof is complete for $k = 1$.

If $k > 1$, we consider two cases, for odd and even n. If n is odd, then we use the same construction of graph G' as in the proof of Theorem 10 to reduce CLIQUE to k-CLIQUE(d) and k-CLUB(d). This is true since $diam(G') \leq k + 1 \leq d$ when k is odd and G has a clique of size m if and only if G' has an k-clique (k-club) of size $m + (\frac{k-1}{2} - 1)|V| + |E| + 1$. If k is even, a similar construction can be used (see Figure 10) to prove the reduction. As before, we use $k/2$ copies of V and a copy of E for the vertex set of the constructed graph $G'' = (V'', E'')$.

$$V'' = \bigcup_{i=1}^{k/2} V^{(i)} \cup E.$$

The edge set E'' is also similar to E' in the previous construction, but instead of connecting vertices from the copy of E to an auxiliary vertex, we make the subset of vertices corresponding to E a clique.

$$\begin{aligned} E'' = & \bigcup_{i=1}^{n/2-1} \{(v^{(i)}, v^{(i+1)}) : v \in V\} \\ & \cup \{(v^{(k/2)}, e) : v \in V, v \text{ is an endpoint of } e \text{ in } G\} \\ & \cup \{(e_1, e_2) : e_1, e_2 \in E, e_1 \neq e_2\}. \end{aligned}$$

Once again, $diam(G'') \leq k + 1 \leq d$ and G has a clique of size m if and only if G'' has an k-clique (k-club) of size $m + (k/2 - 1)|V| + |E|$. This completes the proof of NP-completeness on fixed diameter graphs. □

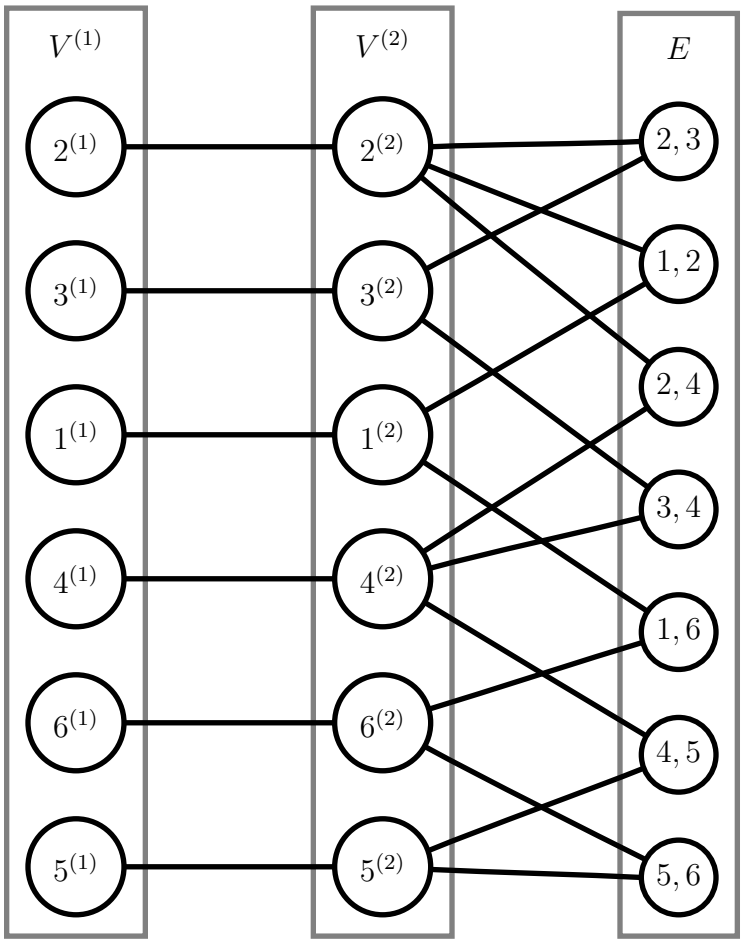


Fig. 10 An illustration to the proof of Theorem 12 for $k = 4$

These complexity results illustrate two important facts. Firstly, these generalizations are hard to solve not only because they generalize cliques, but because they are hard in their own respect (NP-complete for any k). Secondly, the transition in complexity is also sudden, while the problems are easily solved under trivial circumstances when the diameter is bounded above by k, but immediately become NP-complete whenever the diameter of the graph is strictly larger than k.

V.3. Mathematical Programming Formulations

V.3.1. Maximum k-clique Problem

The IP formulation for the maximum k-clique problem on a graph $G = (V, E)$ may be obtained by reducing the problem to the maximum clique problem on the k^{th} power of G, $G^k = (V, E^k)$, where $E^k = \{(i, j) : i, j \in V,\ i < j,\ d_G(i, j) \leq k\}$. G^k is constructed from the original graph by adding edges corresponding to all pairs of vertices with distance no more than k between them in G. Consider the following formulation for k-clique.

$$\begin{array}{rcll} \tilde{\omega}_k(G) & = & \max \sum\limits_{i \in V} x_i, & \\ \text{s. t. :} & & x_i + x_j \leq 1 + \frac{k}{d_G(i,j)} & \forall\ i, j \in V, \\ & & x_i \in \{0, 1\} & \forall\ i \in V. \end{array} \tag{5.3}$$

The constraint ensures that two vertices with $d_G(i, j) > k$ are not simultaneously included in a k-clique, but becomes redundant for pairs of vertices with $d_G(i, j) \leq k$. Since for all pairs of vertices, the shortest path distance is known, the constraint in the system can be replaced by

$$x_i + x_j \leq 1 \ \ \forall\ (i, j) \in \{(i, j) : i, j \in V,\ i < j,\ d_G(i, j) > k\} = \overline{E^k}.$$

The formulation then becomes a maximum clique formulation on G^k. Note that even though the existing heuristics and algorithms for maximum clique problem can be applied to the power of the graph to solve the maximum k-clique problem, their performance may be poorer as the edge density is higher in G^k.

V.3.2. Maximum k-club Problem

The following integer program describes the k-club number.

$$
\begin{aligned}
\bar{\omega}_k(G) \quad &= \quad \max \sum_{i \in V} x_i, \\
\text{s.t.:} \quad & x_i + x_j \leq 1 + \sum_{l: P_{ij}^l \in \mathbb{P}_{ij}} y_{ij}^l \quad \forall\ (i,j) \notin E, \\
& x_p \geq y_{ij}^l \qquad\qquad \forall\ p \in V(P_{ij}^l),\ P_{ij}^l \in \mathbb{P}_{ij},\ (i,j) \notin E, \\
& x_i \in \{0,1\} \qquad\qquad \forall\ i \in V, \\
& y_{ij}^l \in \{0,1\} \qquad\qquad \forall\ P_{ij}^l \in \mathbb{P}_{ij},\ (i,j) \notin E,
\end{aligned}
\tag{5.4}
$$

where $\mathbb{P}_{ij}$ is an indexed collection of all paths of length at most equal to k between vertices i, j in G and P_{ij}^l is the path with index l between vertices i, j. The formulation essentially ensures that if two vertices are in a k-club, then all the vertices in at least one path between them with length less than or equal to k are also included in the k-club. Even though the formulation presented may not be very useful in practice for large values of k, it plays a role in polyhedral study of the problem [16].

V.3.3. Maximum k-plex Problem

The maximum k-plex problem formulation provided here is taken from [15]:

$$
\begin{aligned}
\omega_k(G) \quad &= \quad \max \sum_{i \in V} x_i, \\
\text{s. t. :} \quad & \sum_{j \in V \setminus N[i]} x_j \leq (k-1)x_i + deg_{\bar{G}}(i)(1 - x_i) \quad \forall\ i \in V, \\
& x_i \in \{0,1\} \qquad\qquad \forall\ i \in V.
\end{aligned}
\tag{5.5}
$$

The main constraint here ensures, that if the vertex is in the k-plex ($x_i = 1$), then it has at most $k-1$ non-neighbors inside the k-plex. If the vertex is not in the k-plex, the constraint is redundant. More detail on the mathematical programming formulation of the maximum k-plex problem and its properties may be found in [14, 15].

V.4. Maximum 2-club Problem

An IP Formulation for the maximum 2-club problem can be induced from the formulation for the maximum k-club problem in the previous section. Thus, the 2-club number $\bar{\omega}_2(G)$ of a graph $G = (V, E)$ admits the following integer programming formulation:

$$\begin{aligned} \bar{\omega}_2(G) \quad &= \quad \max \sum_{i \in V} x_i, \\ \text{s.t.} \qquad & x_i + x_j - \sum_{k \in N(i) \cap N(j)} x_k \leq 1 \quad \forall\ (i,j) \notin E, \\ & x_i \in \{0,1\} \qquad\qquad\qquad\qquad \forall\ i \in V. \end{aligned} \tag{5.6}$$

The formulation ensures that if two vertices are in a 2-club and they are not connected, then they have at least one common neighbor inside the 2-club.

A lower bound can be obtained by observing that complete bipartite graphs have diameter 2 and form "edge essential" 2-clubs. That is, if we remove any edge from a complete bipartite graph, its diameter increases to 3. Hence we know that, if the size of the largest complete bipartite subgraph (not necessarily induced) of G is b^*, then $\bar{\omega}_2(G) \geq b^* \geq b \geq \Delta + 1$, where b is the size of a largest known complete bipartite subgraph of G and Δ is the maximum degree. A vertex of maximum degree with its neighbors (*star* subgraphs) is an easy-to-find complete bipartite subgraph of G and hence the bound. The need for b, computed using heuristics or other techniques arises because of the fact that finding b^* is NP-hard [66].

V.5. Numerical Results

For the numerical experiments, two popular protein interaction networks were chosen. The first network is the protein-protein interaction map of the yeast *Saccharomyces Cerevisiae* [89] and the second is the protein-protein interaction map of a gastric

pathogen *Helicobacter Pylori* [34, 120].

As mentioned before, both these networks exhibit power-law degree distribution as shown in Figure 11 and Figure 12. Table 4 and Table 5 contain information on the order and the number of connected components of that order in S. Cerevisiae protein network and H. Pylori protein network respectively. Figure 13 graphically illustrates the protein-protein network of H. Pylori.

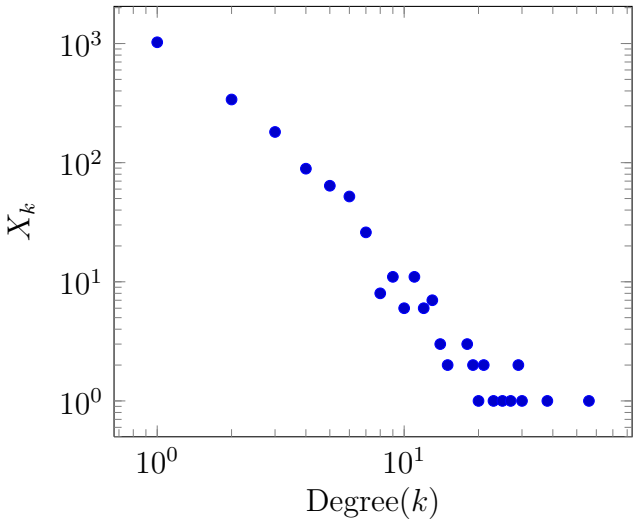


Fig. 11 Degree distribution, in logarithmic scale, for the protein network of S. Cerevisiae, X_k is the number of vertices of degree k

Table 4 S. Cerevisiae. Vertices: 2114; Edges: 2203; Connected components: 417

Order	#Components	Order	#Components
1	268	5	5
2	101	6	3
3	25	7	4
4	10	1458	1

A maximum clique, 2-clique and 2-club were found on both these networks and a maximum 3-clique was found in S. Cerevisiae using exact approaches. Table 6 contains this information. Clique, 2-clique and 3-clique numbers were found by applying

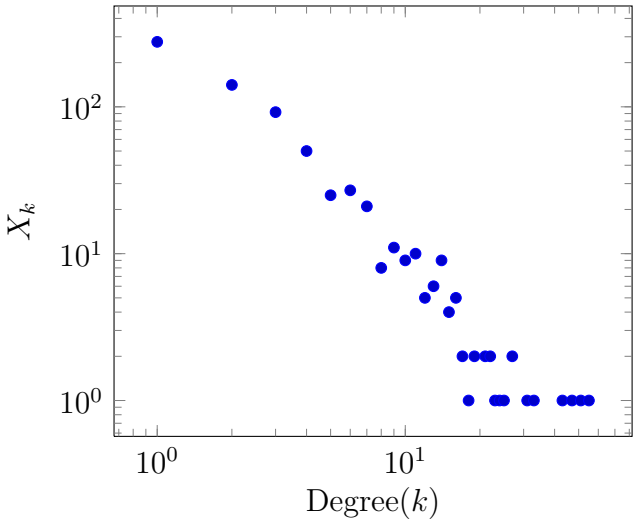


Fig. 12 Degree distribution, in logarithmic scale, for the protein network of H. Pylori, X_k is the number of vertices of degree k

Table 5 H. Pylori. Vertices: 1570; Edges: 1403; Connected components: 858

Order	Number of Components
1	850
2	7
706	1

the Carraghan-Pardalos algorithm [41] for the maximum clique on G, G^2 and G^3, respectively. The 2-club number was found by solving the IP formulation (5.6) using CPLEX® [86]. However, in order to solve the maximum 2-club problem on these graphs, some preprocessing techniques were used to reduce the size of the instances.

Since $\bar{\omega}_2(G)$ is bounded below by $\Delta + 1$, a vertex v such that $|N[N[v]]| < \Delta + 1$ cannot be in any optimal solution and can be removed from the graph. This approach can be used to reduce the size of the instance. The reduced instance was decomposed into subproblems externally by setting $x_v = 1$ for a non-leaf vertex v in the reduced graph and deleting all vertices that are at distance 3 or more from v. We consider only non-leaf vertices, because if a leaf vertex is in a maximum 2-club, then so is its

neighbor. This yields the largest 2-club containing v. Now we can delete v and repeat this process. Algorithm 3 is a pseudo-code summarizing this procedure.

Algorithm 3 Maximum 2-club algorithm

```
1: procedure PreProcessing(G)
2:     Δ ← max{deg_G(v) : v ∈ V
3:     X ← {v ∈ V : |N[N[v]]| < Δ + 1}
4:     if X ≠ ∅ then
5:         G ← G[V \ X]
6:         go to 2
7:     end if
8:     return G
9: end procedure
10:
11: procedure Maximum2Club(G)
12:     G ← PreProcessing(G)
13:     v ← min{i : i ∈ V, deg_G(i) ≥ 2}
14:     x_v = 1
15:     SolveIP(G[N[N[v]]])
16:     G ← PreProcessing(G − v)
17:     if Δ(G) ≥ 2 then
18:         go to 13
19:     end if
20: end procedure
```

In line 15 the IP program (5.6) is formulated for distance-2 closed neighborhood of the chosen vertex v, for which x_v is fixed to be 1, and the problem is solved exactly by CPLEX.

A Pentium® 4 1.4GHz laptop computer was used in the experiments and the run-times were under a minute in cases where the optimum was obtained. In both biological networks, it turned out that the maximum 2-clique and maximum 2-club correspond to the same solution (subset of vertices). Figure 14 and Figure 15 are the maximum 2-clubs (and maximum 2-cliques) of S. Cerevisiae and H. Pylori respectively. Figure 16 shows the maximum 3-clique that was found in S. Cerevisiae. Observe that it is also a (maximum) 3-club, as it basically consists of three star graphs with their central vertices forming a triangle.

Table 6 Clique, 2-Clique, 2-Club, 3-Clique, 3-Club numbers of S. Cerevisiae and H. Pylori protein maps

Network	$\omega(G)$	$\tilde{\omega}_2(G)$	$\bar{\omega}_2(G)$	$\tilde{\omega}_3(G)$	$\bar{\omega}_3(G)$
S. Cerevisiae	6	57	57	68	68
H. Pylori	3	56	56	N/A	N/A

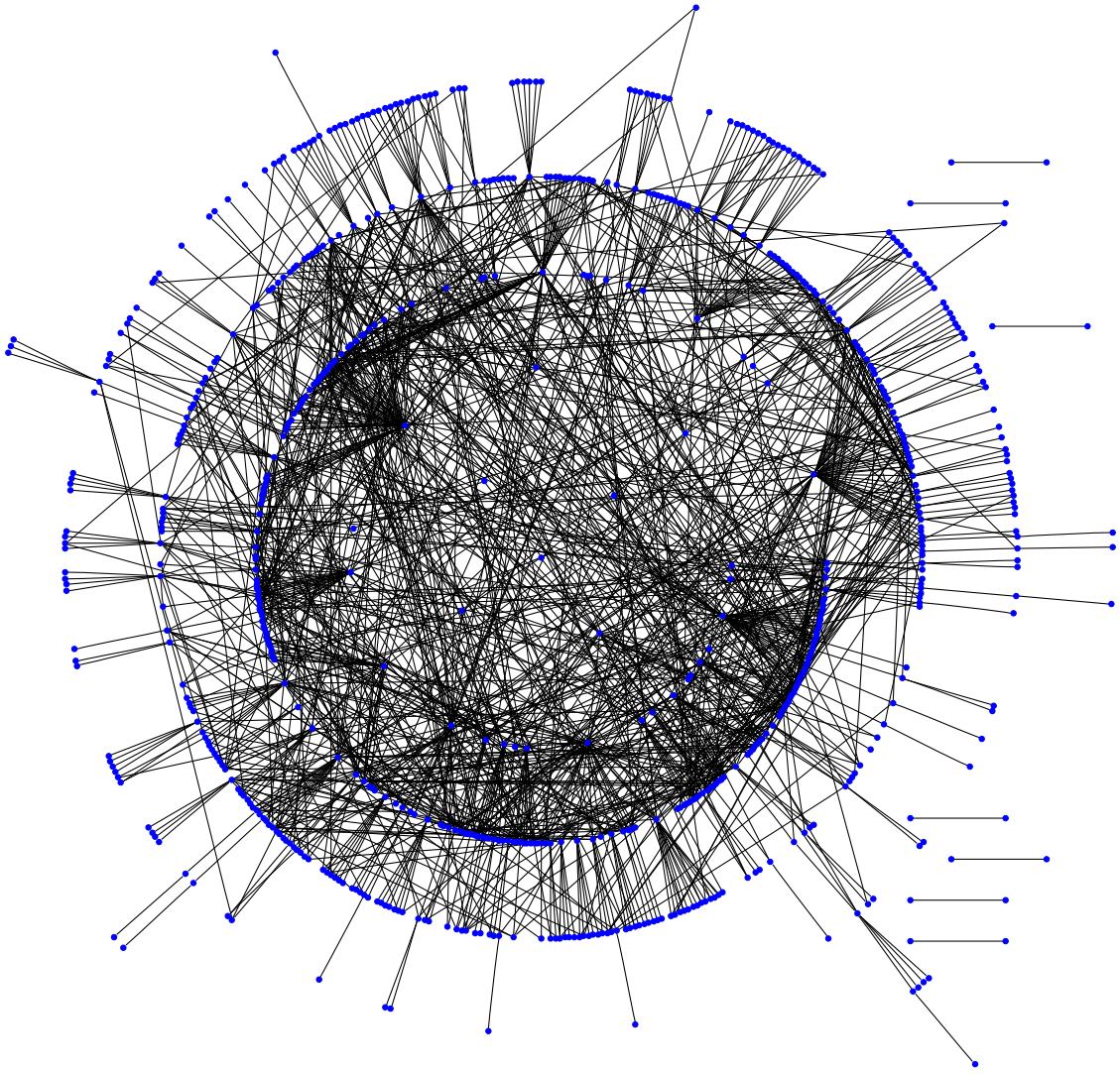


Fig. 13 Protein-protein interaction map of H. Pylori

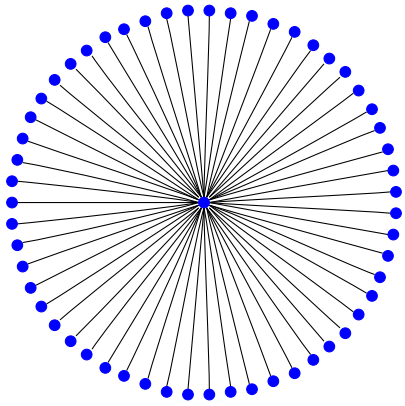


Fig. 14 A maximum 2-club and 2-clique of S. Cerevisiae

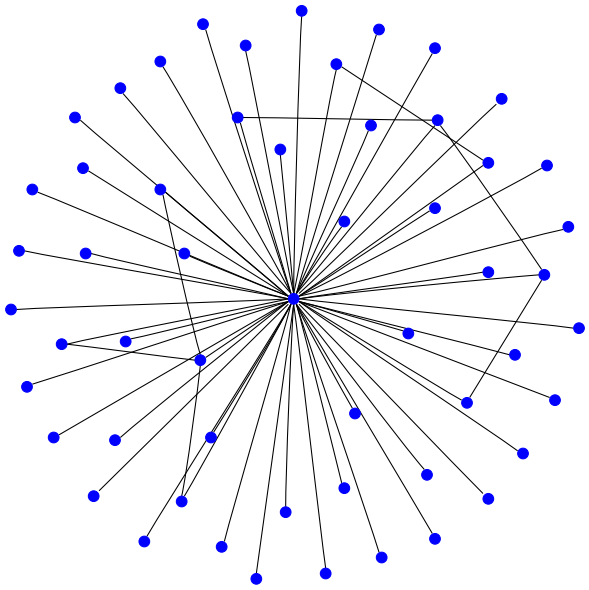


Fig. 15 A maximum 2-club and 2-clique of H. Pylori

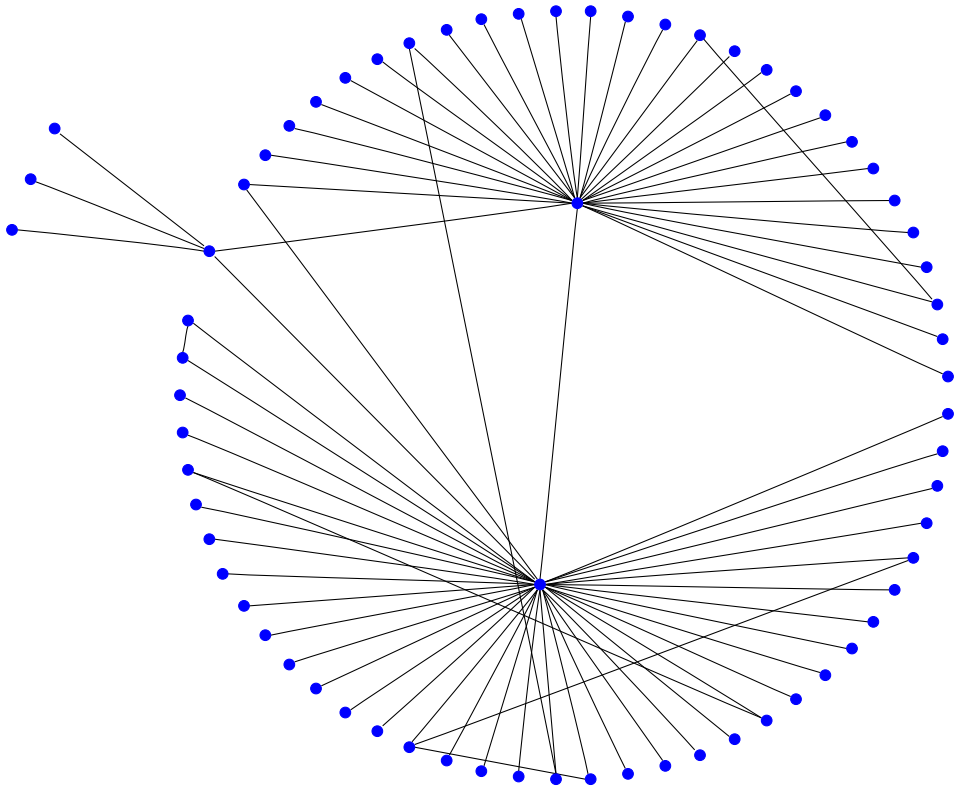


Fig. 16 A maximum 3-clique and 3-club of S. Cerevisiae

CHAPTER VI

EXACT ALGORITHM FOR THE MAXIMUM WEIGHT k-PLEX PROBLEM

This chapter presents the algorithm we developed for the maximum weight k-plex problem. Section VI.1 describes the idea and provides the outline of the algorithm without emphasizing the implementation details. Next sections concentrate with algorithm implementation details, i.e. Section VI.2 discusses the k-plex verification procedure, that is one of the core parts of the algorithm, and Section VI.3 concentrates on the preprocessing part of the algorithm, which is also a crucial point of the algorithm's performance. In Section VI.4, we consider some special cases where the algorithm could be modified to perform better on specific instances. Finally, the algorithm's performance is evaluated by conducting numerical experiments and analyzing the obtained results in Section VI.5 and comparing them with the performance of other existing algorithms in Section VI.6.

VI.1. Algorithm Implementation

Our algorithm for the maximum weight k-plex problem is applicable to a simple undirected graph G with vertex weights given by vector w. We restrict the weight vector to be non-negative. Indeed, a maximum weight k-plex cannot contain a vertex with a negative weight, since removal of such a vertex provides a k-plex with larger weight. The proposed algorithm belongs to the class of exact algorithms, i.e., it provides the exact value of the weighted k-plex number $w_k(G)$ and the corresponding vertex subset that forms a maximum weight k-plex in graph G. The algorithm is based on the idea used by Östergård [111] in his well-known maximum weight clique

detection algorithm. We decided to follow this approach due to the following reasons. Firstly, Östergård algorithm is known as the the state-of-the-art exact algorithm for the maximum weight clique problem. Secondly, the idea can be easily generalized to a wide class of combinatorial optimization problems in graphs. Moreover, the algorithm works with both weighted and unweighted graphs. And, finally, all implementation details, as well as an extensive numerical study is well documented and is available online at the author's website.

Assume that we have a problem of finding a maximum weight subgraph that holds some property Π, and the corresponding number $\nu(G) = \max\{w(P) : \ P \subseteq V,\ G[P]$ holds property $\Pi\}$. An important assumption about the property Π is that it is hereditary on induced subgraph. Recall from chapter II, that if the property Π is *nontrivial*, *interesting* and *hereditary* on the induced subgraphs, then according to Yannakakis [140, 141], the problem of finding a maximum (and maximum weight) induced subgraph with property Π is NP-hard. Generalized algorithm works in the following manner. First, let us fix some order on the vertex set V and use the notation $V = \{v_1, v_2, \cdots, v_n\}$ to refer to the set of ordered vertices. Next, we define the sets $S_i \subseteq V$ as $S_i = \{v_i, v_i + 1, \cdots, v_n\}$. In the algorithm, one needs to compute the function $c(i)$ that is the weight of the maximum induced subgraph with property Π in the induced subgraph $G[S_i]$. Obviously, $c(n) = w(v_n)$ and $c_1 = \nu(G)$, since $S_n = \{v_n\}$ and $S_1 = V$, respectively. The algorithm computes the value of $c(i)$ starting from $c(n)$ and down to $c(1)$. Obviously, $c(i) \geq c(i + 1)$, moreover, if $c(i) > c(i + 1)$, then the subgraph that provides the value of $c(i)$ contains vertex v_i. If we assume that the graph P_i, such that $c(i) = w(P_i)$, does not contain the vertex v_i, then its vertex set $V(P_i) \subseteq S_i \setminus \{v_i\} = S_{i+1}$, so $P_i \subseteq G[S_{i+1}]$, and hence, $c(i + 1) = c(i)$, which

contradicts to the initial condition. Thus, for unweighted graphs

$$c(i) = \begin{cases} c(i+1)+1, & \text{if the corresponding graph contains } v_i \\ c(i+1), & \text{otherwise} \end{cases}$$

For the weighted case it is not possible to claim the same conclusion, but still, in case when $c(i) > c(i+1)$, v_i belongs to the corresponding graph and $c(i) \leq c(i+1)+w(v_i)$. The algorithm computes $c(n), c(n-1), \cdots, c(1)$ using backtracking, and whenever the weight of the current subgraph P and the best possible weight of the remaining part $c(i)$ is less than the best weight weight found so far, we prune.

Algorithm 4 Generalization of Östergård algorithm

```
1: procedure GeneralizedOstergard(G)
2:     Order(V)
3:     max = 0
4:     for i = n downto 1 do
5:         FindMaxPi(S_{i+1}, {v_i})
6:         c(i) = max
7:     end for
8:     return max
9: end procedure
10:
11: procedure FindMaxPi(C,P)
12:     if |C| = 0 then
13:         if w(P) > max then
14:             max = w(P)
15:         end if
16:         return
17:     end if
18:     while C ≠ ∅ do
19:         if w(C) + w(P) < max then
20:             return                                  ▷ Prune point 1
21:         end if
22:         i = min{j : v_j ∈ C}
23:         if c(j) + w(P) < max then
24:             return                                  ▷ Prune point 2
25:         end if
26:         C = C \ {v_i}
27:         P' = P ∪ {v_i}
28:         C' = {v ∈ C : P' ∪ {v} holds property Π}
29:         FindMaxPi(C',P')
30:     end while
31: end procedure
```

Algorithm 4 describes the generalized Östergård algorithm. The main procedure GeneralizedOstergard accepts graph G as the input and calls the recursive procedure FindMaxPi to compute the values of $c(i)$. The procedure FindMaxPi is the core of the algorithm and finds a maximum weight subgraph with property Π using two sets as the input: (i) the *working set* (also known as *candidate list* or *candidate set*) is the set of vertices that may be used to build a subgraph with property Π, and (ii) the set of vertices, that represents the currently found subgraph with property Π. The procedure picks vertices from C one by one, adds a picked vertex to the current graph P, updates the candidate list, and calls itself with new values of input sets. Obviously, if the candidate set is empty, the procedure terminates returning the best weight value found. Other points of termination are the two prune points, which are equivalent to pruning the branch-and-bound tree. Pruning of the first type occurs when the weight of the current subgraph together with the weight of the whole candidate set is less than the best found so far. The second type pruning occurs when the weight of the current subgraph together with $c(i)$ (which is best possible in $G[S_i]$) is less then the best found so far.

The maximum weight k-plex algorithm is obtained from the generalized algorithm by using k-plex specific routines for the candidate list update. Algorithm 5 presented below finds a maximum weight k-plex in given graph G. It will be referred to as the *algorithm* or the *main algorithm* in the remaining part of this chapter.

The following sections concentrate on details of the maximum weight k-plex algorithm implementation, optimization and tuning its configuration settings. In order to evaluate the performance difference caused by changes in the algorithm, we formed a small test-bed from a subset of graphs instances that will be considered in the numerical experiments. We tried to choose graphs that belong to different classes and can be solved within a suitable running time. By a suitable running time we under-

Algorithm 5 Maximum weight k-plex algorithm

```
1: procedure MaxKPlex(G)
2:     Order(V)
3:     max = 0
4:     for i = n downto 1 do
5:         FindKPlex(S_{i+1}, {v_i})
6:         c(i) = max
7:     end for
8:     return max
9: end procedure
10:
11: procedure FindKPlex(C,P)
12:     if |C| = 0 then
13:         if w(P) > max then
14:             max = w(P)
15:         end if
16:         return
17:     end if
18:     while C ≠ ∅ do
19:         if w(C) + w(P) < max then
20:             return                                  ▷ Prune point 1
21:         end if
22:         i = min{j : v_j ∈ C}
23:         if c(j) + w(P) < max then
24:             return                                  ▷ Prune point 2
25:         end if
26:         C = C \ {v_i}
27:         P' = P ∪ {v_i}
28:         C' = {v ∈ C : IsKPlex(P' ∪ {v})}
29:         FindKPlex(C',P')
30:     end while
31: end procedure
```

stand the running time which is not too large, since we need to perform experiments many times for the performance evaluation, but, at the same time, not too small, so that we can easily observe the differences in running time and do not have to deal with hardware clock resolution problem. The test bed consists of two parts, containing weighted and unweighted graphs, respectively. We need to consider both cases, since some of the algorithm's features are applicable to either weighted or unweighted graphs only. The following instances were chosen for the unweighted benchmarks: six instances from DIMACS test bed [54], four instances of Sanchis graphs, used by

Balasundaram [14, pp. 94–96] (two from each of the two classes considered in [14]), and one instance of the market graph (see Chapter VII for a detailed description of the market graph). For the weighted case, the choice of instances is more challenging, since there is no good, widely accepted test set for weighted graph problems. Therefore, we used five instances of the market graph with different order and parameters, and four of the Sanchis graphs used in the unweighted case with randomly generated integer vertex weights, uniformly distributed between 0 and 100. The parameters of interest for this test-bed, such as order, size, edge density and maximum k-plex weight (size for unweighted graphs) for $k = 1, 2, 3, 4, 5$ are presented in Table 9. Note that most of the referenced here, and later in this chapter, tables are located in Appendices due to their large size. Each modification of the algorithm's parameters was evaluated by comparing the running time or other reasonable indicators measured before and after making the modification (with all other algorithm configuration settings unchanged and using exactly the same hardware and software). The results obtained in the process of evaluation of different parameters are not necessarily comparable between themselves, since they could be obtained with different algorithm configuration settings.

VI.2. k-plex Verification Routine

The main difference between the maximum weight clique algorithm and its maximum weight k-plex counterpart is the working set update procedure, which actually mostly determines the running time of the whole algorithm. In case of the maximum clique algorithm, the candidate list is just the intersection of neighborhoods of vertices from the currently constructed clique. The requirement for vertices to belong to all the neighborhoods is a necessary and sufficient condition for them to be in the

candidate list for the maximum weight clique problem algorithm. If the vertex v_i was added at current iteration, then the working set C' for the next iteration is just the intersection of the current working set and the neighborhood of the newly added vertex: $C' = C \cap N(v_i)$. But it is not so easy in the case of k-plex. The working set must be updated explicitly by verifying that each member from the current working set creates a k-plex together with the currently constructed one. This issue caused us to create the k-plex verification function IsKPlex that is used in line 28 of Algorithm 5. The function is called for each members of current working set each time the working set needs to be updated. Table 10 shows the number of IsKPlex calls for the selected graph instances. The speed of IsKPlex determines the speed of working set update process, and, hence, the whole algorithm running time. So, this function is the bottleneck in the algorithm's performance and, thus, there are two ways to improve it: either improve the running time of this function by optimizing it, or reduce the number of calls to this function by changing the outer algorithm structure.

Here we discuss how to optimize the k-plex verification procedure. Formally, IsKPlex is a decision function that accepts a vertex subset $K \subseteq V$ and a positive integer number k and determines whether $G[K]$ is a k-plex. The implementation of this function is trivial: verify that the degree of each vertex from K is at least $|K|-k$, as shown in Algorithm 6:

Algorithm 6 Simple k-plex verification algorithm

```
1: function IsKPlex1(K,k)
2:     for v ∈ K do
3:         if deg_{G[K]}(v) < |K| − k then
4:             return false
5:         end if
6:     end for
7:     return true
8: end function
```

The function IsKPlex1 has a quadratic run time complexity with respect to its argument K size (one loop goes over all vertices explicitly, and a linear-time procedure for determining the degree is called at each loop iteration), and it looks like it is not possible to improve this function in general case. However, it appears that we can improve it by using information already known from previous algorithm's steps. Assume that the vertex u has just been added to the k-plex K at current iteration, so the new k-plex is $K' = K \cup \{u\}$. Then we know that $K \cup \{u\}$ is a k-plex and $K \cup \{v\}$ is a k-plex for any vertex v from C. To determine whether v will be included in C', we need to verify that $K \cup \{u\} \cup \{v\}$ is a k-plex. Let us call a vertex $v \in K$ *saturated* if $|K \setminus N(v)| = |K| - k$, which means that it is not possible to add vertices to K if they are not in $N(v)$ without violating the k-plex requirement. Since $K \cup \{v\}$ is already known to be a k-plex, the only possible violations may be caused by the newly added vertex u. First, by adding u it is possible for some vertex $w \in K$ to increase the number of vertices that w is not connected to. This number may be increased by at most one, and only when the vertex u is not connected to w. If the vertex w becomes saturated, then C' must be a subset of $N(w)$. Second, the vertex u may not be connected to some vertices from K, and thus could be saturated itself, in which case $C' \subseteq N(u)$. All other vertices that are still not saturated after the k-plex extension, do not bring any restrictions to the candidate list. This allows one to create a faster procedure for updating the candidate list. Namely, after adding a new vertex we determine the list of vertices that become saturated, including the newly added one. Then the next iteration candidate list is obtained from the current one by intersecting the current list with the neighborhood of each saturated vertex. In order to find vertices that become saturated fast, we keep and update the list and number of non-neighbors of vertices from K. Algorithm 7 formalizes the incremental k-plex verification procedure.

Algorithm 7 Incremental k-plex verification algorithm

```
Require:
 1: nonneigh[v] - list of non neighbors for vertex v
 2: nncnt[v] - number of non neighbors for vertex v
 3: both arrays are external to this function and are maintained incrementally
 4: At the beginning nonneigh[v] = ∅ and nncnt[v] = 0, ∀v ∈ V
 5:
 6: function MakeSaturatedList(K,k)
 7:     u ← last vertex added to K
 8:     S ← ∅                                          ▷ Saturated vertex list
 9:     for v ∈ K \ {u} do
10:         if (u, v) ∉ E(G) then
11:             nncnt[u] ← nncnt[u] + 1
12:             nncnt[v] ← nncnt[v] + 1
13:             if nncnt[v] = k − 1 then
14:                 S ← S ∪ {v}
15:             end if
16:         end if
17:     end for
18:     if nncnt[u] = k − 1 then
19:         S ← S ∪ {u}
20:     end if
21:     return S
22: end function
23:
24: function IsKplex2(K, k, v)
25:     for u ∈ S do
26:         if (u, v) ∉ E(G) then
27:             return false
28:         end if
29:     end for
30:     return true
31: end function
```

Recall that the aforementioned straightforward verification procedure runs in $O(|K|^2)$ time. The new procedure consists of two parts: generation of the list of saturated vertices, which can be done in $O(|K|)$ time and is performed once, and verification whether a vertex is in the neighborhood of saturated vertices, which is performed for every vertex from the candidate list. The theoretical complexity of the second part is $O(|K|)$, but each vertex can be a member of the saturated vertex list only once during the whole process of building K (once a vertex is included in this list, the candidate list will be intersected with this vertex neighborhood, and no more

vertices from the candidate list may satisfy the condition in line 10 of Algorithm 7). So, practically, the loop in IsKPlex2 is usually empty or performs a small number of iterations.

The comparison of running time of the maximum weight k-plex algorithms, that utilize the original and the alternative functions for the k-plex verification procedure, shows the great improvement in the algorithm's performance, as can be seen from Table 11.

VI.3. Preprocessing Techniques

While the previous section dealt with improvement of the k-plex verification procedure in order to increase algorithm performance, this section investigates whether one can decrease the number of calls to this procedure. Algorithm 5 is a branch and bound algorithm, where isKPlex is called in each branching node many times. It is not possible to just regulate the number of isKPlex calls, since this routine must be called each time when the working set is updated for each vertex from the current working set. So, the possible ways are either to shrink the working set faster (i.e., the branch is cut off because of feasibility), or to prune more often (e.g., the branch is cut off because of the bound). Both goals may be achieved by preprocessing, which is of very different type from the one that was considered in Chapter III. The main difference is that this preprocessing does not reduce the size of the problem, but modifies the order in which the vertices will be considered. The vertex ordering is the main reason of high performance in Östergård's maximum weight clique algorithm. We also try to utilize vertex ordering to improve the performance of the maximum weight k-plex algorithm. Preprocessing based on ordering is a heuristic approach, moreover, it is very different for weighted and unweighted graphs. The simplest ordering, that does

not modify the vertex order at all, serves as the base; and the algorithm's performance depending on different orderings will be evaluated by comparison with the algorithm with no ordering. The following subsections consider the ordering procedures for weighted and unweighted graphs separately.

VI.3.1. Weight Based Ordering

Since we are looking for a k-plex of maximum weight, the natural way to order vertices is according to their weights. The following algorithm presents such an ordering:

Algorithm 8 Weight based vertex ordering

```
1: procedure OrderWeight(V)
2:     U ← ∅                                   ▷ Set U is ordered
3:     for i = |V| downto 1 do
4:         u_i ← arg min{w(v) : v ∈ V}
5:         V ← V \ {u_i}
6:     end for
7:     return U
8: end procedure
```

One may think that the algorithm should start from the vertex of the largest weight, as many greedy approaches do, but this is not the case for the exact algorithms. Plots in Figure 17 show the weight of the best found k-plex and the running time as functions of the number of iterations, when applying the algorithm to one of the Sanchis graphs with 100 vertices and random vertex weights, for both strategies: the smallest to largest weight vertex ordering and the opposite one. As we can see, when started from the vertex of the largest weight, the algorithm finds a large weight k-plex fast, but later is spends much more time than the second strategy. This phenomenon can be explained as follows: the values $c(i)$ and the weight of the best known k-plex W grow very fast in the beginning, but later they cannot grow, since W is already close, or perhaps equal, to the optimal value. Thus, the values of $c(i)$ and W remain equal for a long time, and the algorithm stalls without any improvement.

This leads to the situation, when the pruning in line 24 of Algorithm 5 does not occur, since $c(i) = W$, and that is the reason why the algorithm takes so much time.

Ordering from the smallest to the largest weight benefits from the opposite effect: the values of $c(i)$ are maintained to be as small as possible in order to yield more prunes at point 2, and, thus, despite of a slow start, the algorithm performs much better in general with this ordering strategy than the previous one. Results from Table 12 demonstrate that even after the largest-to-smallest strategy finds the optimal value, it spends approximately the same amount of time additionally to finish the branching, i.e., to prove that the k-plex found is optimal.

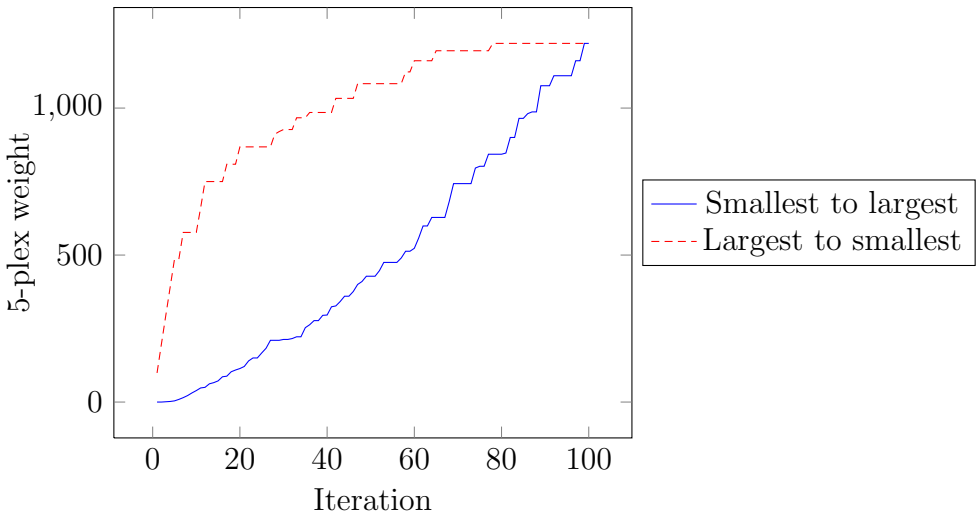
According to the results summarized in Table 13, we made the conclusion, that the preprocessing based on vertex weight ordering significantly improves the performance of the algorithm in the case, when the vertices are ordered from the smallest weight to the largest one, but the algorithm performs very poorly, if the opposite ordering is used.

In the original algorithm for the maximum weight clique, Östergård used the ordering similar to the one considered here, with additional considerations for the situation, when more than one vertices have the same weight.

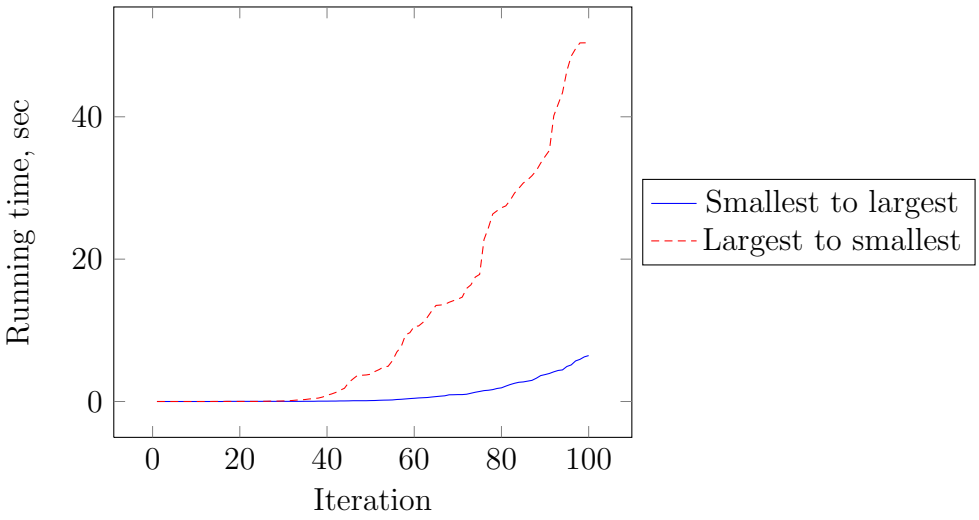
VI.3.2. Degree Based Ordering

When looking for a maximum clique in an unweighted graph, it is natural to order vertices according to their degree. When Östergård's algorithm [111] is applied to unweighted graphs, he performs the vertex ordering based on their degree in the graph induced by remaining vertices (Algorithm 9).

Note, that the resulting set U is formed in reverse order, i.e., the vertices with smallest degree will be the last ones in the ordering. But, since Östergårgd's algorithm starts from the last vertex in the ordered set, the small degree vertices will



(a) 5-plex weight



(b) Running time

Fig. 17 5-plex size and running time for ln2-san100-40w graph

Algorithm 9 Degree based vertex ordering

```
1: procedure OrderOstergard(V)
2:     U ← ∅                                          ▷ Set U is ordered
3:     for i = |V| downto 1 do
4:         u_i ← arg min{deg_{G[V]}(v) : v ∈ V}
5:         V ← V \ {u_i}
6:     end for
7:     return U
8: end procedure
```

be considered before large degree vertices, which is exactly opposite to the simple heuristics for the maximum clique problem, that usually start from a largest degree vertex. The order from smaller degree to larger is supported by the fact that the clique C that contains vertex v is a subset of $N[v]$, so starting with vertices of small degree provides small values for $c(i)$, which yields more cuts at the prune point 2 (Algorithm 4, line 20).

Even though it was created for the maximum clique problem, the degree based ordering performs very well when used with the maximum k-plex algorithm, as was confirmed by numerical experiments (Table 14), and may be justified as follows. On the one hand, since a k-plex is not necessarily a subset of its vertex neighborhood, the ordering is not as justified as for the maximum clique algorithm. On the other hand, a large k-plex induces a small diameter subgraph, and many of its vertices belong to the neighborhood. A natural question arises: how to modify this ordering to cover the whole k-plex and follow exactly the idea used in the maximum clique algorithm? Fortunately, a large order k-plex has diameter at most 2, so it is a subset of $N_2[v]$ for its arbitrary vertex v. A trivial modification in Algorithm 9 provides a new ordering based on the size of the double neighborhood of a vertex instead of its degree.

Numerical comparison between the approaches is presented in Table 15.

Algorithm 10 Double neighborhood based vertex ordering

```
1: procedure OrderOstergard2(V)
2:     U ← ∅                                          ▷ Set U is ordered
3:     for i = |V| downto 1 do
4:         u_i ← arg min{N[N_{G[V]}[v]] : v ∈ V}
5:         V ← V \ {u_i}
6:     end for
7:     return U
8: end procedure
```

VI.3.3. Coloring Based Approach

Another popular vertex ordering for the maximum clique problem emphasizes reducing the working set as fast as possible and is based on the fact that no more than one vertex from an independent set can be included in a clique. The corresponding ordering splits a graph into independent sets (performs graph coloring) and groups vertices of the same color together. Since the problem of graph coloring is NP-hard, the coloring is performed greedily, and the corresponding ordering is computed as shown in Algorithm 11.

Algorithm 11 Coloring based vertex ordering

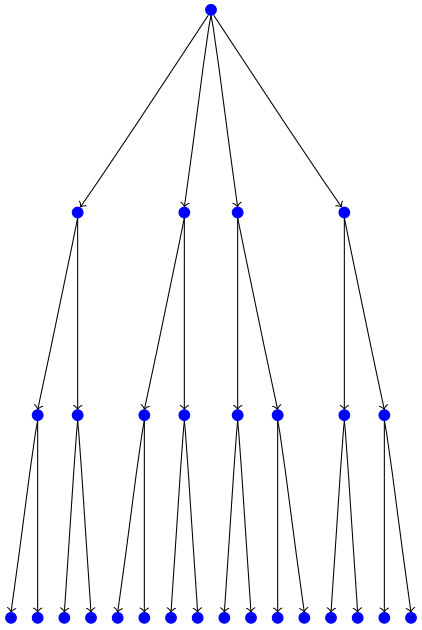
```
1: procedure OrderColoring(V)
2:     U ← ∅                                          ▷ Set U is ordered
3:     col[v] ← 0 ∀v ∈ V
4:     for i = 1 to |V| do                            ▷ Assign the colors
5:         u ← arg min{deg_G(v) : col[v] = 0 and v ∈ V}
6:         col[u] ← min{c ∈ ℕ : c ≠ col[v] ∀v ∈ N(u)}
7:     end for
8:     for j = 1 to max{col[v] : v ∈ V} do
9:         U ← U ⊕ {u : col[u] = j}          ▷ ⊕ means "append to the end"
10:    end for
11:    return U
12: end procedure
```

Obviously, such an ordering is expected to reduce the working set fast, since all vertices of some color will be removed after one vertex of this color will be chosen for inclusion into clique. An interesting question is, in which order to consider the vertices: starting from those, that belong to the color class of the largest cardinality or

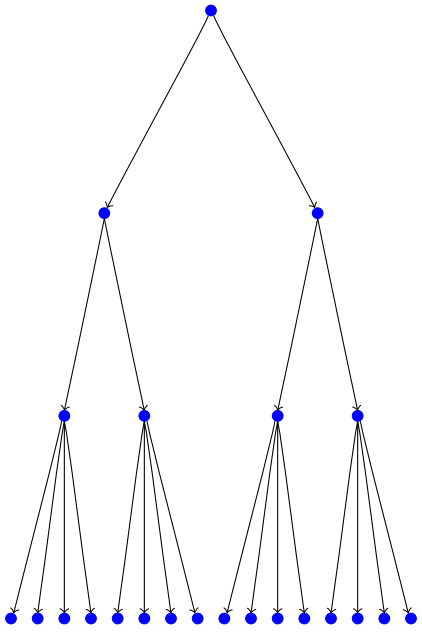
in the opposite order. We may represent the algorithm as a branching tree, where each branch represents a vertex and each color has its level, at which the corresponding vertices are grouped.

Then the first strategy, when the largest group of vertices of the same color is considered first (Figure 18(a)), provides a smaller number of possible choices at the lower level, that are verified with the highest frequency. The second strategy (Figure 18(a)), on contrary, cares more about the upper level of the branching tree and creates less brunches at the upper level, that keeps the whole tree more narrow, but requires more time to consider each node at the lower levels. Since there is no clear indication of which strategy is better, we considered both of them.

When switching from clique to k-plex, the implementation of coloring based ordering also must be changed. First of all, coloring guaranties that only one vertex of each color could be included in a clique. This is not the case for k-plex, since as many as k vertices of the same color may be included. Moreover, there cannot be any property of a vertex set that prevents any k vertices to be included in a k-plex, since any k vertices create a k-plex by definition. In fact, we may still use coloring based ordering for k-plex, but it will be not as efficient for maximum weight k-plex algorithm, as it is for the maximum weight clique algorithm. Another possible choice is to replace the independent set by some other structure, most probably by independent set relaxation, that also limits the number of vertices that may be included together into a k-plex. One possible such structure is the co-k-plex. Balasundaram et al. [15] show that for any co-k-plex, at most $2k - 1 - (1 - (-1)^k)/2$ vertices may be included into a k-plex. It is not known how many vertices from co-m-plex may be included in a k-plex together for arbitrary values of m and k, but obviously for a fixed k this number increases when m is increased. From one side, increasing m creates more branches at each step, but from another side, the number of partitions



(a) Large color first



(b) Small color first

Fig. 18 Branching strategies in maximum weight clique algorithm

in the graph will be smaller. The problem of partitioning a graph into co-k-plexes is known in the literature as defective k-coloring [50, 64] and is NP-hard, so we use a simple greedy procedure (Algorithm 12) to obtain the needed partitions.

Algorithm 12 Defective m-coloring based vertex ordering

```
1: procedure OrderColoring(V, m)
2:     U ← ∅                                              ▷ Set U is ordered
3:     col[v] ← 0 ∀v ∈ V                                  ▷ Color assigned to vertex
4:     colnum[v] ← 0 ∀v ∈ V                ▷ Number of neighbors of the same color
5:     for i = 1 to |V| do                                ▷ Assign the colors
6:         u ← min{v : col[v] = 0 and v ∈ V}
7:         C ← {col[v] : colnum[v] < m − 1 ∀v ∈ N(u)}
8:         if C = ∅ then
9:             col[u] ← max col[v] + 1
10:        else
11:            col[u] ← min C
12:        end if
13:        colnum[u] = |{v ∈ N(u) : col[u] = col[v]}|
14:        colnum[v] = colnum[v] + 1 ∀v ∈ N(u) : col[u] = col[v]
15:    end for
16:    for j = 1 to max{col[v] :  v ∈ V} do
17:        U ← U ⊕ {u : col[u] = j}                ▷ ⊕ means "append to the end"
18:    end for
19:    return U
20: end procedure
```

Experiments were conducted for $m = 1, 2, \cdots, 5$ using both ordering strategies (largest color partition first and the opposite one). Table 16 shows the running time for the largest color first branching strategy and Table 17 presents the running time for the smallest color first strategy.

If the graph is weighted then the next vertex in line 6 of Algorithm 12 is chosen as the smallest weight non-colored vertex. For vertices with the same weight, the one with the largest neighborhood weight is chosen, following the rules similar to the original ordering in [111, 112]. The results for the modified defective coloring based vertex ordering preprocessing are presented in Tables 18 and 19 for both branching strategies.

The main observation from these numerical results is that the preprocessing based

on defective coloring yields better algorithm performance than with no preprocessing at all. As we can see, there is no single answer to the question of which coloring provides the best results. In many cases, the classical coloring (i.e., partitioning the graph into independent sets) provides the best result, but also very often the best result is obtained when $m = k$. Less often, the best running time is given by values of m that are in between 1 and k, and even more rarely, when $m > k$.

In summary, the techniques provided in this section aim to improve the performance of the main algorithm by affecting the branch-and-bound part of the algorithm. The preprocessing techniques based on vertex ordering are heuristic in nature, thus they cannot provide a predictable performance improvement on all problem instances, as approach from Section VI.2. But if they do provide an improvement, it is usually much more significant than that obtained using the techniques from Section VI.2. Finally, the two approaches are not conflicting with each other and can be used together at the same time.

VI.4. Special Cases

VI.4.1. Maximum Weight 2-plex Problem

The idea of incremental k-plex verification presented in Section VI.2 allows one to reduce the running time of the algorithm drastically in most cases, especially for large values of k, but at the same time, the complex and memory-consuming data structure needed to keep the required information, provides some overhead. This includes one-time initial setup, that may cause performance degradation, when the number of verification is relatively small. From Table 11, we can observe that sometimes, when $k = 2$, the running time of the algorithm that used the advanced procedure for k-plex verification is even slightly higher than the running time of the algorithm with the

straightforward verification routine. The described phenomenon does not occur when $k > 2$, most probably since the initial setup overhead does not contribute much to the total algorithm running time.

In order to avoid the degradation of the algorithm's performance when $k = 2$, we created a simpler and better tuned verification routine that works for such case only. If $k = 2$, then the list of possible non-neighbors for a vertex, that is in k-plex, may contain only one element, and thus could be replaced by just a single variable. Also, the number of non-neighbors does not need to be tracked explicitly in this case. Moreover, only one vertex may be saturated at each step. Further, to simplify the algorithm, for each vertex from the working set, we also keep its non-neighbor from the 2-plex already built (if such a non-neighbor exists). So, when adding a new vertex to a 2-plex, it is easy to find a saturated vertex. If a saturated vertex found, it also means that the newly added vertex is also saturated, and the new working set is just an intersection of the current one with the saturated vertex neighborhood and the newly added vertex neighborhood. If there was no saturation, then the non-neighbor values for candidates could be updated depending on whether they are connected to the newly added vertex. The proposed procedure allows one to verify whether a vertex from the candidate list still remain a candidate in constant time. Algorithm 13 demonstrates the described approach:

Table 20 shows the advantage of using the modified routine over the general one for $k = 2$.

VI.4.2. k-plex in Sparse Graphs

The simple fact that for a vertex v from clique C, $C \subseteq N[v]$, allows to reduce the problem of finding the maximum clique from the whole graph to several single-vertex neighborhoods. Utilizing this idea, Carraghan and Pardalos [41] created one of the

Algorithm 13 Incremental 2-plex verification algorithm

```
Require:
 1: nonneigh[v] - non neighbor for vertex v
 2: At the beginning nonneigh[v] = −1, ∀v ∈ V, that means no non-neighbors
 3:
 4: function Is2plex(K, u, v)                    ▷ u - newly added, v - candidate
 5:     if nonneigh[u] ≠ −1 then
 6:         if (u, v) ∈ E(G) and (nonneigh[u], v) ∈ E(G) then
 7:             return true
 8:         else
 9:             return false
10:         end if
11:     else
12:         if (u, v) ∈ E(G) then
13:             return true
14:         else
15:             if nonneigh[v] = −1 then
16:                 nonneigh[v] ← u
17:                 return true
18:             else
19:                 return false
20:             end if
21:         end if
22:     end if
23: end function
```

best known maximum clique algorithms. We used the same idea in Section V.5 to shrink the maximum 2-club finding problem to the induced subgraph of $N_2[v]$, if vertex v is assumed to be in the 2-club. A logical question arises for k-plex. Seidman and Foster [125] showed that if G is a k-plex and $|V(G)| > 2k-2$, then G is connected and $diam(G) \leq 2$. The connectivity of large k-plex has already been used indirectly is Section VI.2, when the candidate list was shrunk by intersection of neighborhoods of saturated vertices, which means that all other candidates to be included in k-plex are connected to the saturated vertex.

The problem with the approach based on limited diameter is that, unlike to the clique and 2-club, the property holds for k-plexes of order at least $2k-1$ only and not necessarily holds for induced subgraphs of G, that are k-plexes too. From the first glance, the limited diameter property cannot be used in the algorithm. Assume that

K is a maximum weight k-plex in G and $|K| \geq 2k-1$, then for any pair of vertices $u, v \in K$, the distance $d_G(u,v) \leq 2$, since $diam(K) \leq 2$. To be mathematically precise, we define the property Π' on the graph K as follows: K is a k-plex in G and $\forall u, v \in K \; d_G(u,v) \leq 2$. The solution to the problem of finding maximum weight set that holds Π' in G equals to the solution of the maximum weight k-plex problem in G, if the cardinality of the solution set is at least $2k-1$. But Π' is hereditary on any induced subgraph and thus can be solved using the generalized framework from VI.1. The only difference between the original maximum k-plex algorithm and the modified one is that after adding new vertex v to the formed k-plex, the working set part outside of $N_2[v]$ is simply cut off, and the problem of finding the required k-plex is reduced to the $N_2[v]$. The only issue about this modified algorithm is that if the resulting k-plex has cardinality smaller than $2k-1$, then the original algorithm must be executed in order to find a proper solution.

The described scale-reduction is the key point in performance improvement, when applied to the sparse graphs with large diameter. Using this technique we were able to find a maximum weight k-plex in instances of the market graph with more than 5000 vertices in Chapter VII. Table 21 shows the advantage of the modified algorithm over the original one.

VI.5. Numerical Experiments

All numerical experiments presented in this chapter were conducted on Dell Optiplex GX620 computer with Intel(R) Pentium(R) D 3.2GHz processor and 2GB of RAM. The algorithm was implemented in C programming language, using Microsoft Visual C++ .NET 2003 (v 7.1) development environment for Win32 platform. Despite of Dual Core feature of CPU, the algorithm is implemented using

single threaded model and cannot use this advantage of hardware.

In these experiments, the algorithm always used the incremental k-plex verification routine, as described in Section VI.2 and the diameter based pruning from Section VI.4.2, since this modification should not degrade the algorithm's performance on any of the considered instances. The special verification routine for 2-plex was not used. In order to decide which vertex ordering to use, we have conducted preliminary experiments for each instance in the following way. The algorithm was executed several times with different preprocessing and ran for only a predefined short amount of time (we limited this execution time to 1 minute), then the progress was compared for different vertex ordering, and the ordering that provides the best progress was chosen for the further benchmarking. The algorithm progress was measured as the number of vertices processed, not as the size of the best found k-plex, due to the specifics of this algorithm.

The test-bed of instances used in our experiments consists of two groups. The first group is comprised of benchmark clique and coloring instances from the Second DIMACS Challenge [54, 91, 131]. Most of the graph from group one are known to be hard instances for the Maximum Clique Problem, so we do not expect them to be easy for the Maximum k-plex Problem either. Instances come from different sources, such as coding theory [35, 37, 99, 127], fault diagnosis [23] and others. For more detail about these instances see [28, 79, 123]. The total number of instances in this group is 88, and we preserve their original names in the table of results. Later we will refer to this group as DIMACS.

The second group consists of graphs, generated by Sanchis generator [123] and used by Balasundaram [14] in his numerical experiments. The Sanchis graphs are known as some of the hardest instances for the Maximum Clique problem [79] and could be generated using given order (n), density (d) and clique number (c). The total

number of graphs generated is 120, half of them has the clique number equal to 1/5 of the total vertex number, while the second half has the clique number approximately equal to $-2\log_d n$. The parameter n was changed from 100 to 1000 with increment interval 100, and parameter d was changed from 0.4 to 0.9 with step size of 0.1. The resulting graphs were named as *prefix*-sanchis-n-d, where *prefix* is "n5" for the first subgroup and "ln2" for the second one, n is the order of the graph and d is its density. Graphs of this group will be referred to as "Sanchis-linear" and "Sanchis-log", following the original [14, pp. 94–96].

In total, we have 208 graphs for this test-bed. Table 22 shows the parameters of the instances: graph order, size, density, clique number and k-plex number for $k = 2, 3, 4, 5$. The clique numbers for DIMACS graphs were obtained either from [54] or from other available sources, while for the Sanchis graphs this number is known in advance. The k-plex numbers were obtained by our algorithm. If the algorithm could not find the optimal solution in a reasonable time, the best found k-plex size is provided, which is the lower bound for the optimal one, so such cases are indicated with $\geq$ in the k-plex number column of the table .

Next, Table 23 provides the running time of the maximum weight k-plex algorithm in CPU seconds. As it was mentioned before, the preprocessing was chosen individually for each instance according to the best performance achieved on preliminary tests. The preprocessing algorithm is specified by its abbreviation in the table. The ordering from the Östergård maximum weight clique algorithm is denoted by O1, the ordering that uses double neighborhood instead of single one (Algorithm 10) is denoted by O2, the degree based ordering is denoted by D, the defective coloring based ordering is denoted by Cm, where m is the parameter of co-m-plex, and finally, the defective coloring based algorithm with weight/degree based greediness is denoted by Wm, where m has exactly the same meaning. The letter R at the end of

abbreviation means that the algorithm started its work in the order that is reverse to the provided by the ordering. Table 7 shows the 10 most frequently used orderings used for each k and 10 most used orderings overall.

Table 7 Top 10 most used orderings

	$k=2$		$k=3$		$k=4$		$k=5$		Total	
N	Ordering	Count	Ordering	Count	Ordering	Count	Ordering	Count	Ordering	Count
1	O1R	104	O1R	112	O1R	91	O1R	89	O1R	396
2	O1	27	C1	24	O1	41	O1	47	O1	137
3	C1	21	O1	22	C1	19	C1	24	C1	88
4	C1R	19	C5R	8	O2	8	O2	7	C1R	32
5	D	9	C3	6	C2R	8	C3R	5	C2R	20
6	C3	4	C1R	5	C4	6	C3	5	C3	19
7	C2R	4	C5	4	C5	5	C2	5	O2	17
8	DR	3	C3R	4	C2	5	C4R	4	C2	17
9	C5R	3	C2R	4	C3	4	C2R	4	D	16
10	C3R	3	C2	4	C1R	4	C1R	4	C3R	15

The most used ordering is the same as used by Östergård in [112] for the maximum clique algorithm, and the second was the ordering from [111] that was used for the maximum weight clique. Coloring and defective coloring perform well in many cases using both strategies. This numerical result confirms the conclusion made before that there is no any single preprocessing technique, that is the best for all instances.

Next group of test instances consists of networks that arise from real-life problems and includes some of the aforementioned networks, in particular Erdös collaboration networks described in Section III.4; protein-protein interaction maps of Saccharomyces Cerevisiae and Helicobacter Pylori used in numerical experiments in Section V.5; college football schedule graph from Chapter I; as well as the *airline networks*, constructed for the 7 major U.S. carriers according to Bureau of Trans-

portation Statistics data for July 2005. In the airline network, vertices represent airports and connections correspond to the airline's flight between these airports. These networks are named as NN.NET, where NN is a two-letter code representing an airline. Finally, we consider ALL7.NET graph, which is basically a union of airline network for all 7 airlines; SKYTEAM.NET is the airline network for SkyTeam; and USAIR97.NET is obtained from [20] and models all US airports and all regular air flights between them as of 1997.

Table 24 presents the parameters and the maximum k-plexes order for these networks. It is easy to observe, that all these networks are very sparse and their maximum k-plexes are rather small. The same approach as for previous groups was used to determine the best preprocessing ordering. Optimal solution was found in all instances of this group. The algorithm running times are presented in Table 25. Interestingly, the simple ordering based on vertex degree dominates in this group, which may be explained in the following way. The running time of the main algorithm is determined by graph density, while the running time of the preprocessing ordering is mostly determined by the number of vertices in the graph. Compared to the Dimacs and Sanchis graphs, the graphs of current group are sparser, so the main algorithm performs better, but at the same time, the order of the graph is larger, causing the preprocessing to take more time. So, the ratio between the preprocessing running time and the main algorithm running time for this group is much larger than for Dimacs and Sanchis graphs, and the preprocessing running time may be a significant part of the total running time. This is the reason why the fastest of the proprocessing procedures was chosen during the preliminary run.

Finally, we applied the algorithms to the weighted graphs. We used instances of the market graph generated for one-year time periods for years 1990 to 2006 using zero correlation threshold, see Section VII.2 for more detail. Due to the large size of

these instances, we are not able to find the maximum weight k-plex for $k > 2$. The graph parameters, maximum weight clique and 2-plex size and weight, as well as CPU time are provided in Table 28. Note that the maximum weight clique or k-plex is not necessarily the largest one by cardinality, moreover, the cardinality of the maximum weight 2-plex may be even less than the cardinality of the maximum weight clique. The common tendency is that for the market graph the coloring based preprocessing starting in reverse order is the best one in most of the cases. This fact supports the hypothesis about the similarity of the structure of these graph instances and may serve as motivation for developing a specialized maximum weight k-plex algorithm for the market graph.

VI.6. Comparison with Existing Approaches

The introduction of the maximum weight k-plex problem to the operations research community by Balasundaram at el. [15] attracted a lot of interest and triggered the development of several algorithms for solving the problem. As a result, at least four approaches (including the original one and the current one) were presented in the last two years.

The first approach was introduced in [14], where Balasundaram developed a brunch-and-cut approach for the maximum 2-plex problem and conducted numerical experiments for a subset of DIMACS, Sanchis-linear and Sanchis-log groups. Two versions of the branch-and-cut algorithm for the maximum k-plex problem were implemented. Both versions employ CPLEX for solving the IP formulation of the problem, the difference between versions being in the cuts used. One version incorporates the maximum independent set cuts (referred as BC-MIS), while the other incorporates co-k-plex cuts (referred as BC-C2PLX). For more detail about these

approaches see [14, pp. 98–99].

The author has not considered the weighted version of the problem, so we provide the comparison for the unweighted case. Table 27 shows the running time of the three approaches: BC-MIS and BC-C2PLX by Balasundaram and our algorithm (referred as T), conducted on exactly the same hardware system. In most cases, our algorithm outperforms both branch-and-cut approaches, which is expected due to optimization of the algorithm to the concrete problem, while branch-and-cut is a more general approach. Also, higher running times result from using an IP solver that is universal and is not optimized for the concrete problem.

McClosky [103, 104] considered two different ways of finding a maximum k-plex in a graph. One way is based on Carraghan-Pardalos [41] ideas, while the second one is based on the aforementioned Östergård's idea. Both algorithms were tested on a subset of DIMACS benchmarks. Since the second of his approaches outperforms the first one on all test instances, we will make comparison with the second approach only. In his experiments, McClosky limited the algorithm execution time to one hour, so we also disregard all results that were obtained in a larger amount of time. The numerical results were provided for $k = 2, 3, 4$ with precision 1 sec, the running time 0 in McClosky experiments was interpreted as <1sec. Running time comparison was done only for those cases where the algorithms terminated at the optimum. Table 26 provides the running times, where the columns marked with M refer to McClosky's experiments and those marked by T show our results. From the results, we conclude that our algorithm demonstrated better running time, perhaps due to applying the preprocessing technique.

Finally, Wu and Pei [139] developed a parallel algorithm for enumeration of all the maximal k-plexes in the graph. Since they enumerate all maximal k-plexes, they are also able to find the maximum one. However, their work emphasizes the par-

allelization of the algorithm and comparison between serial and parallel algorithms performance. They do not provide any numerical results for the well-known benchmark instances, and we cannot make any comparison with our approach.

CHAPTER VII

PORTFOLIO SELECTION VIA IDENTIFYING WEIGHTED k-PLEXES IN FINANCIAL NETWORKS

In this chapter, we introduce a new approach to selecting robust diversified portfolios by modeling the stock market as a network and utilizing combinatorial optimization techniques proposed in this dissertation to identify maximum weight k-plexes in the obtained networks. The proposed approach is based on the *market graph* model, which was previously used for efficient grouping of stocks according to certain correlation-based criteria. The proposed techniques provide a new framework for *diversified portfolio selection*, which results in robust portfolios that consistently outperform the market trends. Section VII.1 provides stock market essentials. Then in Section VII.2 we demonstrate how to apply the k-plex approach to the stock market. Finally, Section VII.3 provides numerical results, comparison with previously used model and concludes the study.

VII.1. Introduction

Making efficient and profitable investments in the highly volatile conditions of the modern stock market is a challenging and exciting problem. Many strategies of restricting the potential risks of investments deal with the well-known concept of *diversified portfolios*. In general, in a diversified portfolio, the pairwise correlation between the behavior of different stocks in the portfolio does not exceed a certain (low) correlation threshold. The goal of this approach is to ensure that the behavior of the portfolio is more robust with respect to potential downswings of the entire market. In addition, the most important issue in selecting a good diversified port-

folio is its overall profitability. Therefore, the main challenge in diversified portfolio selection is finding a balance between the goals that often contradict to each other: the *profitability* and the *robustness* of the portfolio.

Portfolio selection techniques have a long history, dating back to the classical work of Markowitz [102]. Since then, a lot of mathematical modeling techniques were developed for the analysis of the stock market and portfolio selection. However, a relatively new technique for visualization, information retrieval, and portfolio selection in the stock market that utilizes the concepts from graph theory has been introduced very recently, and its potential in terms of diversified portfolio selection has been indicated. Boginski et al. [25–27] have introduced the network-based model of the stock market (referred to as the *market graph*), which was constructed based on the information on pairwise correlations between all stocks traded in the U.S. stock markets over a specified period of time. One of the main challenge that authors dealt with is that the large number of stocks on the market, that is increasing over the time (Figure 19). A number of interesting results regarding the global properties of this graph and their dynamics have been obtained by Boginski et al., including the important fact that this graph has a *power-law* degree distribution, which is common in a number of other real-world networks arising in sociology, biology, telecommunications, etc. Moreover, it has been indicated that cliques and independent sets in this graph can be used for tackling clustering problems in the stock market; specifically, identifying groups of similar and dissimilar stocks according to the natural criterion of pairwise correlation thresholds. In particular, independent sets, which represent groups of vertices with no connections, were utilized to find "perfectly diversified" portfolios, where the correlation between *each* pair of stocks did not exceed a certain low threshold value. This innovative approach was the first application of graph theory in the context of portfolio selection; however, its main disadvantage was the fact

that the profitability of the portfolios was not explicitly taken into account. In this study, we make the further step to extend this graph theoretic methodology that will allow selecting robust diversified portfolios that have superior profitability.

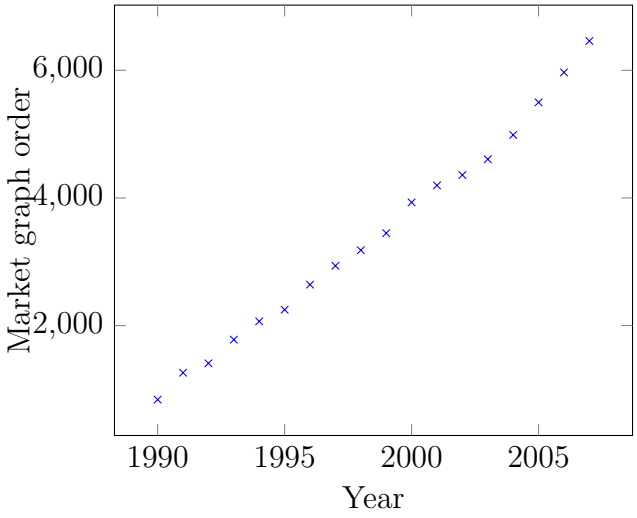


Fig. 19 Market graph order over time

The idea behind the proposed approach is to utilize the introduced graph theoretic concepts of *clique relaxations in weighted graphs*, where the weight of each vertex (stock) corresponds to its return over the considered time period, and each pair of vertices is connected by an edge if the correlation between the corresponding pair of stocks does not exceed a certain threshold. More specifically, the problems of finding maximum weight cliques and maximum weight k-plexes in the market graph will be considered. The optimal solutions of these combinatorial optimization problems will be analyzed from the profitability and robustness perspectives.

VII.2. Problem Setup

VII.2.1. Constructing the Market Graph

The main concept utilized in this study is the *market graph* originally proposed in [25], which essentially is a network-based representation of the entire U.S. stock market. In this network, each stock traded in the U.S. stock markets is represented by a vertex. For all pairs of vertices (i, j), $i = 1, \ldots, N$, $j = 1, \ldots, N$, where N is the total number of stocks, the correlation coefficient C_{ij} is calculated using the following simple procedure.

Assume that $P_i(t)$ is the price of stock i on day t. Then $R_i(t) = \ln \frac{P_i(t)}{P_i(t-1)}$ is defined as the logarithm of return of the stock i from day $(t-1)$ to day t [100]. Then, the correlation coefficient between instruments i and j is

$$C_{ij} = \frac{\langle R_i R_j \rangle - \langle R_i \rangle \langle R_j \rangle}{\sqrt{\langle R_i^2 - \langle R_i \rangle^2 \rangle \langle R_j^2 - \langle R_j \rangle^2 \rangle}}, \tag{7.1}$$

where $\langle R_i \rangle$ is the mean of $R_i(t)$ over N trading days, that is, $\langle R_i \rangle = \frac{1}{N} \sum_{t=1}^{N} R_i(t))$.

Further, in the context of the considered problem, an edge (i, j) will exist between vertices i and j if the corresponding correlation coefficient $C_{ij} \leq \theta$, where θ is the correlation threshold value, which can be changed to construct different instances of the market graph. Note that a complementary graph, where an edge (i, j) would be placed if $C_{ij} > \theta$, can also be constructed. Figure 20 shows such an example of market graph, built for ten randomly chosen stocks with correlation threshold $\theta = 0.05$.

It is also worth mentioning that the parameter θ, which can be controlled by the user, can be treated as a quantitative measure of the degree of diversification, or intra-portfolio correlation.

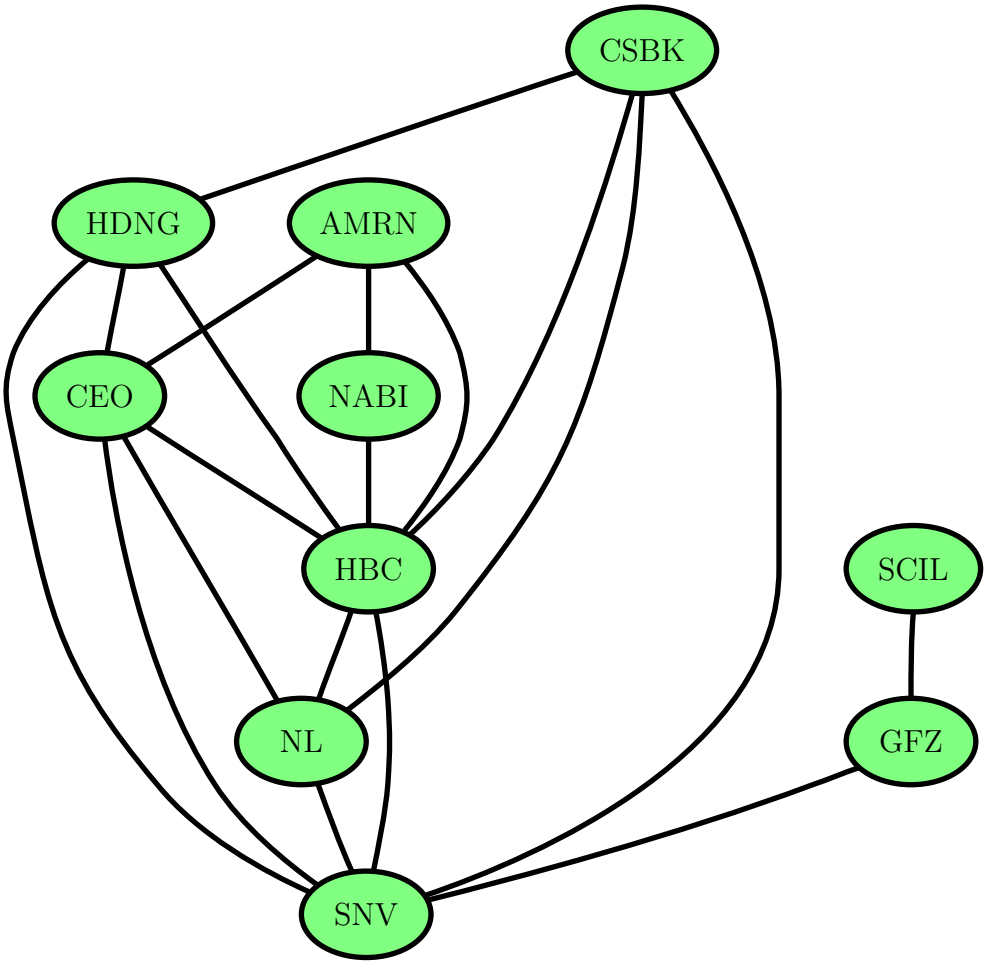


Fig. 20 Market graph for 10 randomly chosen stocks

VII.2.2. Weighted Market Graphs

Cliques in the market graphs constructed using the aforementioned procedure correspond to independent sets in the graphs considered by Boginski et al. in [25–27]. In these studies, it has been indicated that finding independent sets (i.e., cliques in the present settings) in the market graph represents a promising innovative approach to selecting diversified portfolios in the market. Moreover, it has been shown that for any given stock, it is possible to identify a diversified portfolio containing this stock, where the size of these portfolios is determined primarily by the value of the correla-

tion threshold θ. However, the main drawback of this approach is the fact that the *returns* of the identified diversified portfolios were not explicitly taken into account.

In this work, we address this issue and propose an efficient model for selecting high-return diversified portfolios using a modified *weighted market graph* model, where each vertex is assigned a weight, which represents the return of the corresponding stock over the considered time period. To calculate the weight w_i of each vertex i, the following definitions can be used:

- $w_i^1 = \log \frac{P_i(N)}{P_i(1)}$,
- $w_i^2 = \frac{P_i(N)}{P_i(1)} - 1$,

which essentially characterize the overall return of each stock i over the entire considered period of N trading days.

We proceed with constructing the weighted market graphs corresponding to 500-day trading periods based on the stock prices data between 1990 and 2006. Using the concepts of weighted cliques and weighted clique relaxations, such as weighted k-plexes described above, we can solve the corresponding combinatorial optimization problems on the constructed graph instances.

It should be noted that weighted clique relaxations have certain advantages compared to weighted cliques in the context of portfolio selection. In particular, weighted 2-plexes appear to be appropriate network structures that can be considered in these settings. The reasoning behind this idea is the fact that cliques tend to be overly restrictive models of diversified portfolios, where every two stocks in the portfolio need to be uncorrelated, which makes it challenging to find large diversified portfolios modeled as cliques in the market graph, as indicated in [27]. On the other hand, reasonably "tight" clique relaxations, such as 2-plexes provide a good balance between the quality and the size of the identified diversified portfolios.

Next, we can compare the optimal solutions of different optimization problems that can be formulated in the considered context and draw conclusions regarding the practical efficiency of the proposed approaches. In the following subsection, we present and analyze the results of these computational experiments.

VII.3. Computational Experiments

Taking into account the arguments mentioned above, we formulated and solved different optimization problems for several instances of the market graph corresponding to different time periods. Similarly to the computations performed in [27], each of the considered time periods consisted of 500 trading days over approximately 2 years. The solutions of the following optimization problems were considered and compared:

- maximum (unweighted) clique problem (similar to [27]);
- maximum weight clique problem;
- maximum weight 2-plex problem.

In our numerical results we used the second definition of weight (w_i^2), since it exactly expresses the rate of return. The stock price data, collected from public sources is not perfect, so we do not consider the stocks that have missed more than 10% of price values. For the stocks, that have at least 90% of prices we used linear interpolation to recover missed data. Moreover, stocks that have large price fluctuations, like 50% over year period, were also considered, as outliers and were not included in the resulting market graph.

After solving these optimization problems, we investigated the optimal solutions in terms of weighted diversified portfolios corresponding to different time periods. In

the case of the maximum unweighted clique problem, the total weight of the identified portfolio was calculated after the optimal solution was found.

Table 8 presents the results corresponding to 500-day trading periods between 1990 and 2006. Not surprisingly, the total weight of the diversified portfolios found by solving the maximum unweighted clique problem is inferior to the solutions of the maximum weight clique and maximum weight 2-plex problems. However, an interesting observation is that the optimal solutions of the maximum weight 2-plex problem consistently outperform the ones corresponding to the maximum weight clique problem in terms of both the size and the total weight of the portfolios. Clearly, both of these characteristics of the selected portfolios are important from the investor's point of view, since in addition to the overall profitability, a good quality diversified portfolio needs to contain a relatively large number of stocks to provide better "robustness" properties. In addition, one can observe that in the first and the fourth considered time periods, the portfolios selected by solving the maximum weight 2-plex problem also outperform the weighted clique counterparts in terms of the ratio of the total weight to the total number of stocks (i.e., weight per stock in the portfolio).

These results suggest that solving the maximum weight 2-plex problem is an appropriate and promising technique for portfolio selection, since the obtained results exhibit attractive performance characteristics in all the considered recent time periods.

In this study, we proposed an innovative and promising method for tackling a challenging problem of selecting diversified portfolios in the modern stock market. The graph theoretic concepts of clique relaxations that have been studied from theoretical and algorithmic points of view in this dissertation were successfully applied for the first time in the context of financial markets. Computational results presented in this study are encouraging in terms of the performance of selected portfolios over all

Table 8 Parameters of calculated weighted diversified portfolios corresponding to 500-day trading periods

		Portfolio size			Portfolio weight		
Period	N	Clique	Weighted Clique	Weighted 2-plex	Clique	Weighted Clique	Weighted 2-plex
1990–1991	867	10	7	8	2.78	8.43	10.57
1991–1992	1265	12	8	10	2.88	14.83	19.86
1992–1993	1421	12	9	10	2.12	14.51	18.64
1993–1994	1787	12	9	10	1.65	12.12	15.82
1994–1995	2080	13	8	11	1.67	14.48	18.76
1995–1996	2256	13	11	11	5.58	15.21	18.37
1996–1997	2658	13	9	12	2.38	14.66	17.51
1997–1998	2962	14	9	9	0.95	10.02	15.25
1998–1999	3200	13	8	9	-0.04	12.23	16.76
1999–2000	3479	13	8	10	0.48	13.76	17.13
2000–2001	3943	13	9	9	2.90	16.15	21.05
2001–2002	4208	13	8	11	0.87	15.42	19.15
2002–2003	4379	13	9	12	5.48	19.32	22.76
2003–2004	4630	13	10	12	6.86	21.48	25.37
2004–2005	5025	13	10	10	0.96	16.19	20.37
2005–2006	5516	15	10	11	-0.87	15.84	19.05

the considered time periods.

We believe that this approach can be successfully used in practice. Moreover, the results presented in this study along with the previous work presented in [25–27] suggest that applications of advanced graph theoretic techniques in quantitative finance are worth investigating in-depth in a variety of related problems.

CHAPTER VIII

CONCLUSION AND FUTURE WORK

This dissertation mostly concentrates on solving NP-hard problems on graphs, particularly, on the MWISP and the maximization problems for the clique relaxation models.

For the first problem we developed a new scale-reduction approach based on critical weight set. The proposed approach should be used as a preprocessing phase for the exact algorithm, and allows to reduce the size of the original problem by fixing some graph vertices to be present or absent in the final optimal solution. Since the problem of finding a critical weight set is easy to solve, the approach does not require as much running time as the main algorithm, and using the critical set approach before applying the main algorithm results in faster computation than applying the exact algorithm directly to the original problem instance. The numerical experiments confirmed the theoretical results. We have also studied the relationship between our approach and other preprocessing approaches used for the MWISP based on very different techniques, but providing similar results in the end.

As for future work in this direction, an interesting question is whether it is possible to extend the critical weight approach to the relaxations of the independent set, in particular to the co-k-plex. First of all, similarly to the critical weighted independent set, we may define the critical weighted co-k-plex in the following way: for a graph $G = (V, E)$, let $\mathcal{I}_k$ denote the set of all co-k-plexes of G. Then I_c is a critical weight co-k-plex if it maximizes the difference $w(I) - w(N(I))$ over all co-k-plexes in the graph G (over set $\mathcal{I}_k$). The corresponding critical numbers will be

defined as $\alpha_c^{(k)}(G) = \max\{w(I) - w(N(I)) : \ \forall I \in \mathcal{I}_k\}$. Since

$$\mathcal{I}_1 \subseteq \mathcal{I}_2 \subseteq \cdots \subseteq \mathcal{I}_k \subseteq \cdots \subseteq 2^V,$$

we have

$$\alpha_c(G) = \alpha_c^{(1)}(G) \leq \alpha_c^{(2)}(G) \leq \cdots \leq \alpha_c^{(k)}(G) \leq \cdots \leq \mu(G).$$

Recalling from Section III.1 that $\alpha_c(G) = \mu(G)$, we obtain:

$$\alpha_c(G) = \alpha_c^{(1)}(G) = \alpha_c^{(2)}(G) = \cdots = \alpha_c^{(k)}(G) = \cdots = \mu(G).$$

Thus, the problem of finding a critical weight co-k-plex in the graph is polynomially solvable. Moreover, a critical weight independent set is an optimal solution of the critical weight co-k-plex problem for any k. This fact provides the hope that the critical set approach may be used for the maximum weight co-k-plex problem, but the logical question is how to find a more useful solution for the critical weight co-k-plex set, i.e., a critical weight co-k-plex that is not an independent set. We think that the answer to this question may be obtained by extending the work by Larsen [95], that allows one to find the maximum weight critical weight independent set in polynomial time. Finally, a major open question in this approach concerns with obtaining results similar to Lemma 3 and Theorem 4 from Section III.2, that allow to develop the scale-reduction algorithm. More generally, as it was mentioned in [140], if property Π is non-trivial, interesting and hereditary on induced subgraphs, then either all independent sets or all cliques satisfy Π. If cliques satisfies property Π, but independent sets do not, then we may switch to the problem of finding $\bar{\Pi}$ on $\bar{G}$ and all independent sets satisfy $\bar{\Pi}$. So, without loss of generality, from all properties Π we consider only those that are satisfied by all independent sets and define $\alpha_c^{(\pi)}(G) = \max\{w(I) - w(N(I)) : I \subseteq V$ and I holds $\Pi\}$. Then $\alpha_c(G) = \alpha_c^{(\pi)}(G) = \mu(G)$, and the problem of finding

a critical weight induced subgraph with property Π is solvable in polynomial time. The same questions as for the critical weight co-k-plex problem arise: how to find a non-trivial solution and is it possible to utilize the critical weight set with property Π for finding a maximum weight induced subgraph with property Π?

If such a method is developed, then we may think about approaches from Chapter IV with respect to the introduced properties Π, in particular with respect to co-k-plex. It is of interest to investigate the optimal solution of the integer programming formulation for the maximum co-k-plex problem and to determine whether it has the structure similar to that from [110]; formulate the problem as a quadratic unconstrained pseudo-boolean problem and apply the roof-duality approach to this formulation in order to find the possible weak or strong persistence. As it was already mentioned, the roof-duality may be applied to any quadratic unconstrained binary problem, but it does not guarantee that scale-reduction will be obtained. So, the hardest point to find is an appropriate problem formulation. Finally, it is of interest to consider a structure, that extends the concept of t-hat to the co-k-plex, and verify whether the result similar to the one obtained in Theorem 8 of Section IV.8 still holds.

For the second group of problems, we first needed to perform a mathematical study of these problems. It has not been done before, since the clique relaxation models were considered in social network studies and were not studied well from the mathematical point of view. We considered three possible clique relaxations: k-clique, k-club, k-plex, and formulated the corresponding optimization problems. Also, we proved, that all three problems are NP-hard and presented new mathematical programming formulations.

There are still many open questions concerning the clique relaxation models that should be addressed in future work. First, it would be interesting to analyze the complexity of the k-clique, k-club and k-plex (co-k-plex) problems on some restricted

graph classes, such as bipartite or planar graphs. Another complexity related question deals with the issue of gap recognition between $\omega_k(G)$ and $\omega_m(G)$, and may be formulated as follows. Given a graph G and fixed positive integers k and m, does G has the same weight for maximum k-plex and m-plex? Most likely, the problem in NP-hard, since the same question for the k-club and m-club (k-clique and m-clique) has been recently answered in [38] and the corresponding problem has been determined to be NP-hard. Another question mentioned in Section VI.3 is the following: what is the order of a maximum k-plex in a co-m-plex? The answers for $m = 1$ and $m = k$ were given by Balasundaram et al. [15], but are still unknown for other values of m. One may try to generalize the result from Theorem 9 that established the relation between $\omega(G)$ and $\omega_k(G)$, in order to find the relation between $\omega_k(G)$ and $\omega_m(G)$. Finally, it is of interest to find a quadratic unconstrained binary formulation for the problems and apply the roof-duality approach.

In this dissertation we developed the algorithm for the maximum weight k-plex problem and studied its performance issues in details. The efficiency of the developed algorithm was confirmed by conducting extensive computational experiments and, also, by comparing the obtained results with results for existing approaches. The algorithm provided for the maximum weight k-plex problem is not only important for this problem solution, but also provides a general framework for developing similar algorithms for many other problems.

The possible future extensions of this algorithm can be seen at several different levels. On the highest and most theoretical level, one can study the necessity of the key requirement for this algorithm, that is heredity on induced subgraphs requirement for property Π, which makes the problem of finding a maximum induced subgraph with property Π NP-hard, according to the work by Yannakakis [140]. Since later, in [141], the author considered the effect of connectivity for this problem class and stated the

similar result when additional requirement of connectivity is applied to the property Π, an interest point is to investigate the effect of connectivity on the algorithm and to determine if it is possible to modify Algorithm 4 in order to find the maximum weight connected induced subgraph with property Π. At the middle level, since the generalization of the Östergård algorithm for the maximum weight clique problem may be applied to any problem of finding a maximum weight induced subgraph that holds property Π, when property Π is hereditary on induced subgraphs, it is natural to apply this algorithm to different possible Π. The general scheme provides an algorithm for solving the corresponding problem, however, one needs to provide an effective routine for updating the working set and an effective ordering based preprocessing to ensure a good algorithm performance. One may also investigate different possible preprocessing strategies that may improve the algorithm even for the maximum weight k-plex problem. Finally, a good software programming exercise is to develop an efficient data structure for initial graph and temporary data representation in order to improve the running time of the candidate list updating routine for the maximum weight k-plex algorithm, and to confirm that "coefficient does matter" when solving engineering problems.

Finally, considering other clique relaxation models that were not covered in this dissertation provides a wide area of research opportunities.

REFERENCES

1. Aarts, E., Korst, J.: Simulated Annealing and Boltzmann Machines. John Wiley & Sons Incorporated, Chichester, UK (1989)

2. Aarts, E., Lenstra, J.K.: Local Search in Combinatorial Optimization. John Wiley and Sons, New York, NY (1997)

3. Abello, J., Pardalos, P., Resende, M.G.C.: On maximum clique problems in very large graphs. In: J. Abello, J. Vitter (eds.) External Memory Algorithms, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, chap. 50, pp. 119–130. American Mathematical Society, Providence, RI (1999)

4. Abello, J., Resende, M.G.C., Sudarsky, S.: Massive quasi-clique detection. In: S. Rajsbaum (ed.) LATIN 2002: Theoretical Informatics, pp. 598–612. Springer-Verlag, London (2002)

5. Ageev, A.A.: On finding critical independent and vertex sets. SIAM Journal on Discrete Mathematics **7**(2), 293–295 (1994)

6. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: Data Structures and Algorithms, 1st edn. Addison-Wesley Publishing Company, Boston, MA (1974)

7. Alba, R.D.: A graph-theoretic definition of a sociometric clique. Journal of Mathematical Sociology **3**, 113–126 (1973)

8. Almaas, E., Barabási, A.L.: Power laws in biological networks. In: E. Koonin (ed.) Power Laws, Scale-Free Networks and Genome Biology, pp. 1–11. Springer U.S., New York, NY (2006)

9. Anderson, T., Mahajan, R., Spring, N., Wetherall, D.: Rocketfuel: An ISP Topology Mapping Engine. Available at http://www.cs.washington.edu/research/networking/rocketfuel/. Accessed June 2008

10. Arora, S.: Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. Journal of the ACM **45**(5), 753–782 (1998)

11. Arora, S., Safra, S.: Approximating clique is NP-complete. In: Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science, Pittsburgh, PA, October 24–27, 1992, pp. 2–13. IEEE Computer Society Press, Piscataway, NJ (1992)

12. Arquilla, J., Ronfeldt, D.: What next for networks and netwars? In: J. Arquilla, D. Ronfeldt (eds.) Networks and Netwars: The Future of Terror, Crime, and Militancy, pp. 311–361. RAND Corporation, Santa Monica, CA (2001)

13. Bader, J.S., Chaudhuri, A., Rothberg, J.M., Chant, J.: Gaining confidence in high-throughput protein interaction networks. Nature Biotechnology **22**(1), 78–85 (2004)

14. Balasundaram, B.: Graph theoretic generalization of clique: Optimization and extensions. Ph.D. dissertation, Texas A&M University, College Station, TX (2007)

15. Balasundaram, B., Butenko, S., Hicks, I.V., Sachdeva, S.: Clique relaxations in social network analysis: The maximum k-plex problem. (2008). Submitted to Operations Research

16. Balasundaram, B., Butenko, S., Trukhanov, S.: Novel approaches for analyzing biological networks. Journal of Combinatorial Optimization **10**(1), 23–39 (2005)

17. Balinski, M.L.: On a selection problem. Management Science **17**(3), 230–231 (1970)

18. Barabási, A.L., Albert, R.: Emergence of scaling in random networks. Science **286**, 509–512 (1999)

19. Barabási, A.L., Jeong, H., Neda, Z., Ravasz, E., Schubert, A., Vicsek, T.: Evolution of the social network of scientific collaborations. PHYSICA A **311**, 3 (2002)

20. Batagelj, V.: Networks/Pajek graph files. Available at http://vlado.fmf.uni-lj.si/pub/networks/pajek/data/gphs.htm (2005). Accessed June 2008

21. Battiti, R., Protasi, M.: Reactive local search for the maximum clique problem. Algorithmica **29**(4), 610–637 (2001)

22. Bazaraa, M.S., Jarvis, J.J., Sherali, H.D.: Linear Programming and Network Flows, 3rd edn. Wiley-Interscience [John Wiley & Sons], Hoboken, NJ (2005)

23. Berman, P., Pelc, A.: Distributed probabilistic fault diagnosis for multiprocessor systems. In: Proceedings of the 20th Annual International Symposium on Fault-Tolerant Computing, Newcastle, UK, June 26–28, 1990, pp. 340–346. IEEE Computer Society Press, Los Alamitos, CA (1990)

24. Biggs, N.L., Lloyd, E.K., Wilson, R.J.: Graph Theory 1736–1936. Clarendon Press, Oxford (1976)

25. Boginski, V., Butenko, S., Pardalos, P.: On structural properties of the market graph. In: A. Nagurney (ed.) Innovations in Financial and Economic Networks, pp. 29–45. Edward Elgar Publishing, Northampton, MA (2003)

26. Boginski, V., Butenko, S., Pardalos, P.: Statistical analysis of financial networks. Computational Statistics & Data Analysis **48**(2), 431–443 (2005)

27. Boginski, V., Butenko, S., Pardalos, P.: Mining market data: A network approach. Computers and Operations Research **33**(11), 3171–3184 (2006)

28. Bomze, I.M., Budinich, M., Pardalos, P.M., Pelillo, M.: The maximum clique problem. In: D.Z. Du, P.M. Pardalos (eds.) Handbook of Combinatorial Optimization, vol. 4, pp. 1–74. Kluwer Academic Publishers, Dordrecht, The Netherlands (1999)

29. Bomze, I.M., Budinich, M., Pelillo, M., Rossi, C.: A new "annealed" heuristic for the maximum clique problem. In: P.M. Pardalos (ed.) Approximation and Complexity in Numerical Optimization: Continuous and Discrete Problems, pp. 78–96. Kluwer Academic Publishers, Dordrecht, The Netherlands (2000)

30. Bomze, I.M., Budinich, M., Pelillo, M., Rossi, C.: Annealed replication: A new heuristic for the maximum clique problem. Discrete Applied Mathematics **121**(1–3), 27–49 (2002)

31. Bomze, I.M., Pelillo, M., Giacomini, R.: Evolutionary approach to the maximum clique problem: Empirical evidence on a larger scale. In: I.M. Bomze, T. Csendes, R. Horst, P.M. Pardalos (eds.) Developments of Global Optimization, pp. 95–108. Kluwer Academic Publishers, Dordrecht, The Netherlands (1997)

32. Boros, E., Hammer, P.L.: Pseudo-boolean optimization. Discrete Applied Mathematics **123**, 155–225 (2002)

33. Boros, E., Hammer, P.L., Tavares, G.: Preprocessing of unconstrained quadratic binary optimization. Available at http://rutcor.rutgers.edu/pub/rrr/reports2006/10_2006.pdf, Rutgers University (2006). Accessed June 2008

34. Biomolecular Relations in Information Transmission and Expression. Generalized protein interactions. KEGG BRITE Database. Available at http://www.genome.jp/brite/generalized_interactions.html (2003). Accessed June 2008

35. Brouwer, A., Shearer, J., Sloane, N., Smith, W.: A new table of constant weight codes. IEEE Transactions on Information Theory **36**(6), 1334–1380 (1990)

36. Busygin, S., Butenko, S., Pardalos, P.M.: A heuristic for the maximum independent set problem based on optimization of a quadratic over a sphere. Journal of Combinatorial Optimization **6**(3), 287–297 (2002)

37. Butenko, S., Pardalos, P.M., Sergienko, I.V., Shylo, V., Stetsyuk, P.: Estimating the size of correcting codes using extremal graph problems. In: C. Pearce (ed.) Optimization: Structure and Applications. Kluwer Academic Publishers, Dordrecht, The Netherlands (2003)

38. Butenko, S., Prokopiev, O.: On k-club and k-clique numbers in graphs. (2008). Submitted to Operations Research

39. Butenko, S., Trukhanov, S.: Using critical sets for the maximum independent set problem solving. Operations Research Letters **35**(4), 519–524 (2007)

40. Butenko, S., Wilhelm, W.E.: Clique-detection models in computational biochemistry and genomics. European Journal of Operational Research **127**(1), 1–17 (2006)

41. Carraghan, R., Pardalos, P.: An exact algorithm for the maximum clique problem. Operations Research Letters **9**, 375–382 (1990)

42. Chen, H., Chung, W., Xu, J.J., Wang, G., Qin, Y., Chau, M.: Crime data mining: A general framework and some examples. Computer **37**(4), 50–56 (2004)

43. Chen, J., Zhang, F.: Extended Nemhauser-Trotter Theorem and A new measure for the vertex cover problem (2006). Dept. of Computer Science, Texas A&M University, College Station, TX. Unpublished

44. Chesler, E.J., Langston, M.A.: Combinatorial genetic regulatory network analysis tools for high throughput transcriptomic data, Subseries of Lecture Notes in Computer Science. In: S. Istrail, P. Pevzner, M. Waterman (eds.) Systems Biology and Regulatory Genomics, chap. 16, pp. 150–165. Springer, Berlin, Germany (2006)

45. Cheswick, W.: Internet Mapping Project. Available at http://www.cheswick.com/ches/map/. Accessed June 2008

46. Chlebík, M., Chlebíková, J.: Crown reductions for the minimum weighted vertex cover problem. Discrete Applied Mathematics **156**(3), 292–312 (2008)

47. Chvátal, V.: Linear Programming. A Series of Books in the Mathematical Sciences. W. H. Freeman and Company, New York, NY (1983)

48. Cook, D.J., Holder, L.B.: Graph-based data mining. IEEE Intelligent Systems **15**(2), 32–41 (2000)

49. Cook, W., Cunningham, W., Pulleyblank, W., Schrijver, A.: Combinatorial Optimization. John Wiley and Sons, New York, NY (1998)

50. Cowen, L., Goddard, W., Jesurum, C.E.: Defective coloring revisited. Journal of Graph Theory **24**(3), 205–219 (1997)

51. Dantzig, G.B.: Origins of the simplex method. In: S.G. Nash (ed.) A History of Scientific Computing, ACM Press History Series, pp. 141–151. Association for Computing Machinery, New York, NY (1990)

52. Davis, R.H.: Social network analysis: An aid in conspiracy investigations. FBI Law Enforcement Bulletin pp. 11–19 (1981)

53. Diestel, R.: Graph Theory. Springer-Verlag, Berlin, Germany (1997)

54. DIMACS: Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge. ftp://dimacs.rutgers.edu/pub/challenge/ (1995). Accessed June 2008

55. Euler, L.: Solutio problematis an geometrian situs pertinentis. Commentarii Academiae Scientiarum Imperialis Petropolitanae **8**, 128–140 (1736). Available at http://math.dartmouth.edu/~euler/docs/originals/E053.pdf, Accessed June 2008

56. Fadiel, A., Langston, M.A., Peng, X., Perkins, A.D., Taylor, H.S., Tuncalp, O., Vitello, D., Pevsner, P., Naftolin, F.: Computational analysis of mass spectrometry data using novel combinatorial methods. In: AICCSA '06: Proceedings

of the IEEE International Conference on Computer Systems and Applications, Dubai/Sharjah, UAE, March 8–11, 2006, pp. 266–273. IEEE Computer Society Press, Washington, DC (2006)

57. Fauske, K.M.: A graphviz to LaTeX converter. Available at http://www.fauskes.net/code/dot2tex/ (2006). Accessed June 2008

58. Feo, T.A., Resende, M.G.C.: A greedy randomized adaptive search procedure for maximum independent set. Operations Research **42**, 860–878 (1994)

59. Feo, T.A., Resende, M.G.C.: Greedy randomized adaptive search procedures. Journal of Global Optimization **6**, 109–133 (1995)

60. Feuersänger, C.: Manual for package pgfplots. Available at http://www.ctan.org/tex-archive/help/Catalogue/entries/pgfplots.html. Accessed June 2008

61. Fischer, I., Meinl, T.: Graph based molecular data mining - an overview. In: W. Thissen, P. Wieringa, M. Pantic, M. Ludema (eds.) Systems, Man and Cybernetics, 2004 IEEE International Conference Proceedings, Den Haag, The Netherlands, October 10–13, 2004, vol. 5, pp. 4578–4582. IEEE Computer Society Press, Washington, DC (2004)

62. Fowler, P.A.: The Königsberg bridges–250 years later. The American Mathematical Monthly **95**(1), 42–43 (1988)

63. Friden, C., Hertz, A., de Werra, D.: STABULUS: A technique for finding stable sets in large graphs with tabu search. Computing **42**(1), 35–44 (1989)

64. Frik, M.: A survey of (m, k)-colorings. In: J. Gimbel, J.W. Kennedy, L.V. Quintas (eds.) Quo Vadis, Graph Theory?, Annals of Discrete Mathematics, vol. 55, pp. 45–58. Elsevier Science Publishers, New York, NY (1993)

65. Gagneur, J., Krause, R., Bouwmeester, T., Casari, G.: Modular decomposition of protein-protein interaction networks. Genome Biology **5**(8), R57.1–R57.12 (2004)

66. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-completeness. W.H. Freeman and Company, New York, NY (1979)

67. Garfinkel, R.S., Nemhauser, G.: Integer Programming. John Wiley and Sons, New York, NY (1972)

68. Gibbons, L.E., Hearn, D.W., Pardalos, P.M., Ramana, M.V.: Continuous characterizations of the maximum clique problem. Mathematics of Operations Research **22**(3), 754–768 (1997)

69. Glover, F.: Tabu search - part I. ORSA Journal on Computing **1**(3), 190–260 (1989)

70. Glover, F.: Tabu search - part II. ORSA Journal on Computing **2**(1), 4–32 (1990)

71. Glover, F., Laguna, M.: Tabu search. In: D. Du, P.M. Pardalos (eds.) Handbook of Combinatorial Optimization, vol. 3, pp. 621–757. Kluwer Academic Publishers, Boston, MA (1998)

72. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Publishing Company, Boston, MA (1989)

73. Gomory, R.E.: Outline of an algorithm for integer solutions to linear programs. Bulletin of the American Mathematical Society **64**, 275–278 (1958)

74. Goodman, S.E., Hedetniemi, S.T.: Introduction to the Design and Analysis of Algorithms. McGraw-Hill, New York, NY (1977)

75. Graphviz - Graph visualization software. Available at http://www.graphviz.org/About.php. Accessed June 2008

76. Grossman, J., Ion, P., Castro, R.D.: The Erdös Number Project. Available at http://www.oakland.edu/enp/ (2007). Accessed June 2008

77. Hammer, P.L., Hansen, P., Simeone, B.: Roof duality, complementation and persistency in quadratic 0-1 optimization. Mathematical Programming **28**(2), 121–155 (1984)

78. Harary, F., Ross, I.C.: A procedure for clique detection using the group matrix. Sociometry **20**(3), 205–215 (1957)

79. Hasselberg, J., Pardalos, P., Vairaktarakis, G.: Test case generators and computational results for the maximum clique problem. Journal of Global Optimization **3**(4), 463–482 (1993)

80. Hayes, B.: Graph theory in practice: Part I. American Scientist **88**(1), 9–13 (2000)

81. Hayes, B.: Graph theory in practice: Part II. American Scientist **88**(1), 104–109 (2000)

82. Hifi, M.: A genetic algorithm-based heuristic for solving the weighted maximum independent set and some equivalent problems. Journal of the Operational Research Society **48**(6), 612–622 (1997)

83. Hochbaum, D.S.: Approximation Algorithms for NP-hard Problems. PWS Publishing Company, Boston, MA (1997)

84. Hopfield, J.J.: Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences **79**, 2554–2558 (1982)

85. Hunt III, H.B., Marathe, M.V., Radhakrishnan, V., Ravi, S.S., Rosenkrantz, D.J., Stearns, R.E.: NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs. Journal of Algorithms **26**(2), 238–274 (1998)

86. ILOG: CPLEX 9.0 Reference Manual. ILOG CPLEX Division, Sunnyvale, CA (2003)

87. Jagota, A.: Approximating maximum clique with a Hopfield network. IEEE Transactions on Neural Networks **6**(3), 724–735 (1995)

88. Jagota, A., Sanchis, L., Ganesan, R.: Approximately solving maximum clique using neural networks and related heuristics. In: D.S. Johnson, M.A. Trick (eds.) Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, DIMACS Series, vol. 26, pp. 169–204. American Mathematical Society, Providence, RI (1996)

89. Jeong, H., Mason, S.P., Barabási, A.L., Oltvai, Z.N.: Centrality and lethality of protein networks. Nature **411**, 41–42 (2001)

90. Jiang, D., Tang, C., Zhang, A.: Cluster analysis for gene expression data: A survey. IEEE Transactions on Knowledge and Data Engineering **16**(11), 1370–1386 (2004)

91. Johnson, D.S., Trick, M.A. (eds.): Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, DIMACS Series, vol. 26. American Mathematical Society, Providence, RI (1996)

92. Kirkpatrick, S., Gellat, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**, 671–680 (1983)

93. Krishna, P., Vaidya, N., Chatterjee, M., Pradhan, D.: A cluster-based approach for routing in dynamic networks. ACM SIGCOMM Computer Communication Review **27**(2), 49–64 (1997)

94. Land, A.H., Doig, A.G.: An automatic method of solving discrete programming problems. Econometrica **28**(3), 497–520 (1960)

95. Larsen, C.E.: A note on critical independence reductions. Available at http://math.uh.edu/~clarson/larson-critical-independence-revised.pdf, University of Houston (2006). Accessed June 2008

96. Lawler, E.L., Wood, D.E.: Branch-and-bound methods: A survey. Operations Research **14**(4), 699–719 (1966)

97. Luce, R.D.: Connectivity and generalized cliques in sociometric group structure. Psychometrika **15**(2), 169–190 (1950)

98. Luce, R.D., Perry, A.D.: A method of matrix analysis of group structure. Psychometrika **14**(2), 95–116 (1949)

99. MacWilliams, F.J., Sloane, N.J.A.: The Theory of Error Correcting Codes. North-Holland, Amsterdam, The Netherlands (1979)

100. Mantegna, R.N., Stanley, H.E.: An Introduction to Econophysics: Correlations and Complexity in Finance. Cambridge University Press, Cambridge, UK (2000)

101. Marchiori, E.: Genetic, iterated and multistart local search for the maximum clique problem. In: Applications of Evolutionary Computing, Lecture Notes in Computer Science, vol. 2279, pp. 112–121. Springer-Verlag, Berlin, Germany (2002)

102. Markowitz, H.M.: Portfolio selection. Journal of Finance **7**, 77–91 (1952)

103. McClosky, B.: Independence systems and stable set relaxations. Ph.D. dissertation, Rice University, Houston, TX (2008)

104. McClosky, B., Hicks, I.V.: Detecting cohesive subgraphs. Available at http://www.caam.rice.edu/~bjm4/paper3.pdf (2008). Accessed June 2008

105. Milgram, S.: The small world problem. Psychology Today **1**(1), 61–67 (1967)

106. Minty, G.J.: On maximal independent sets of vertices in claw free graphs. Journal of Combinatorial Theory, Series B **28**, 284–304 (1980)

107. Mokken, R.J.: Cliques, clubs and clans. Quality and Quantity **13**, 161–173 (1979)

108. Motzkin, T.S., Straus, E.G.: Maxima for graphs and a new proof of a theorem of Turán. Canadian Journal of Mathematics **17**, 533–540 (1965)

109. Nemhauser, G., Wolsey, L.: Integer and Combinatorial Optimization. John Wiley and Sons, New York, NY (1988)

110. Nemhauser, G.L., Trotter, L.E.: Vertex packings: Structural properties and algorithms. Mathematical Programming **8**(2), 232–248 (1975)

111. Östergård, P.R.J.: A new algorithm for the maximum-weight clique problem. Electronic Notes in Discrete Mathematics **3**, 153–156 (1999)

112. Östergård, P.R.J.: A fast algorithm for the maximum clique problem. Discrete Applied Mathematics **120**(1-3), 197–207 (2002)

113. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley, Boston, MA (1994)

114. Papadimitriou, C.H., Vempala, S.: On the approximability of the traveling salesman problem. In: Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, Portland, OR, May 21–23, 2000, pp. 126–133. ACM Press, New York, NY (2000)

115. Pardalos, P.M., Phillips, A.T.: A global optimization approach for solving the maximum clique problem. International Journal of Computer Mathematics **33**, 209–216 (1990)

116. Pardalos, P.M., Prokopyev, O.A., Shylo, O.V., Shylo, V.P.: Global equilibrium search applied to the unconstrained binary quadratic optimization problem. Optimization Methods & Software **23**(1), 129–140 (2008)

117. Pei, J., Jiang, D., Zhang, A.: On mining cross-graph quasi-cliques. In: KDD '05: Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, Chicago, IL, August 21–24, 2005, pp. 228–238. ACM Press, New York, NY (2005)

118. Pelillo, M.: Heuristics for maximum clique and independent set. In: C.A. Floudas, P.M. Pardalos (eds.) Encyclopedia of Optimization, vol. 2, pp. 411–423. Kluwer Academic Publisher, Boston, MA (2001)

119. Peng, X., Langston, M.A., Saxton, A.M., Baldwin, N.E., Snoddy, J.R.: Detecting network motifs in gene co-expression networks. In: Proceedings of the International Conference for the Critical Assessment of Microarray Data Analysis (CAMDA 2004), Lecture Notes in Computer Science. Springer (2004)

120. Rain, J.C., Selig, L., Reuse, H.D., Battaglia, V., Reverdy, C., Simon, S., Lenzen, G., Petel, F., Wojcik, J., Schachter, V., Chemama, Y., Labigne, A., Legrain, P.: The protein-protein interaction map of helicobacter pylori. Nature **409**(6817), 211–215 (2004)

121. Resende, M.G.C.: An optimizer in the telecommunications industry. SIAM Activity Group on Optimization Views-and-News **18**(2), 8–19 (2007)

122. Rhys, J.M.W.: A selection problem of shared fixed costs and network flows. Management Science **17**(3), 200–207 (1970)

123. Sanchis, L., Jagota, A.: Some experimental and theoretical results on test case generators for the maximum clique problem. Available at http://dimacs.rutgers.edu/TechnicalReports/abstracts/1993/93-69.html 69, DIMACS (1993). Accessed June 2008

124. Schrijver, A.: On the history of combinatorial optimization (till 1960). In: K. Aardal, G.L. Nemhauser, R. Weismantel (eds.) Handbook of Discrete Optimization, pp. 1–68. Elsevier, Amsterdam, The Netherlands (2005)

125. Seidman, S.B., Foster, B.L.: A graph theoretic generalization of the clique concept. Journal of Mathematical Sociology **6**, 139–154 (1978)

126. Shilo, V.P.: The method of global equilibrium search. Cybernetics and Systems Analysis **35**(1), 68–74 (1999)

127. Sloane, N.J.A.: On single-deletion-correcting codes. In: K.T. Arasu, A. Seress (eds.) Codes and Designs, Ohio State University Mathematical Research Institute Publications, vol. 10, pp. 273–291. Walter de Gruyter, Berlin, Germany (2002)

128. Soriano, P., Gendreau, M.: Tabu search algorithms for the maximum clique problem. In: Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, DIMACS Series, vol. 26, pp. 221–242. American Mathematical Society, Providence, RI (1996)

129. Spirin, V., Mirny, L.A.: Protein complexes and functional modules in molecular networks. Proceedings of the National Academy of Sciences **100**(21), 12,123–12,128 (2003)

130. Terveen, L., Hill, W., Amento, B.: Constructing, organizing, and visualizing collections of topically related web resources. ACM Transactions on Computer-Human Interaction **6**(1), 67–94 (1999)

131. Trick, M.: Graph coloring and its generalizations. Available at http://mat.gsia.cmu.edu/COLOR04/ (2005). Accessed June 2008

132. Vazirani, V.: Approximation Algorithms, 1st edn. Springer-Verlag, New York, NY (2003)

133. Warrier, D., Wilhelm, W.E., Warren, J.S., Hicks, I.V.: A branch-and-price approach for the maximum weight independent set problem. Networks **46**(4), 198–209 (2005)

134. Washio, T., Motoda, H.: State of the art of graph-based data mining. SIGKDD Explorations Newsletter **5**(1), 59–68 (2003)

135. Wasserman, S., Faust, K.: Social Network Analysis: Methods and Applications. Cambridge University Press, Cambridge, UK (1994)

136. Watts, D., Strogatz, S.: Collective dynamics of "small-world" networks. Nature **393**, 440–442 (1998)

137. Wilhelm, W.E.: A technical review of column generation in integer programming. Optimization and Engineering. International Multidisciplinary Journal to Promote Optimization Theory & Applications in Engineering Sciences **2**(2), 159–200 (2001)

138. Wolsey, L.: Integer Programming. John Wiley and Sons, New York, NY (1998)

139. Wu, B., Pei, X.: A parallel algorithm for enumerating all the maximal k-plexes. In: J.G. Carbonell, J. Siekmann (eds.) Emerging Technologies in Knowledge Discovery and Data Mining, Lecture Notes in Computer Science, chap. 47, pp. 476–483. Springer, Berlin, Germany (2007)

140. Yannakakis, M.: Node-and edge-deletion NP-complete problems. In: STOC '78: Proceedings of the 10th Annual ACM Symposium on Theory of Computing, San Diego, CA, May 01–03, 1978, pp. 253–264. ACM Press, New York, NY (1978)

141. Yannakakis, M.: The effect of a connectivity requirement on the complexity of maximum subgraph problems. Journal of the ACM **26**(4), 618–630 (1979)

142. Zhang, C.Q.: Finding critical independent sets and critical vertex subsets are polynomial problems. SIAM Journal on Discrete Mathematics **3**(3), 431–438 (1990)

143. Zhang, X.S.: Neural Networks in Optimization. Kluwer Academic Publishers, Dordrecht, The Netherlands (2000)

144. Zhang, Y., Abu-Khzam, F.N., Baldwin, N.E., Chesler, E.J., Langston, M.A., Samatova, N.F.: Genome-scale computational approaches to memory-intensive applications in systems biology. In: SC '05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, Seattle, WA, November 12–18, 2005, p. 12. IEEE Computer Society Press, Washington, DC (2005)

APPENDIX A

EXACT k-PLEX ALGORITHM NUMERICAL STUDY

Table 9 Graph parameters for small test-bed

Graph	$\|V\|$	$\|E\|$	Density, %	Maximum k-plex weight				
				$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
DSJR500.1.col	500	3555	2.85	12	14	15	15	16
MANN_a9.clq	45	918	92.73	16	26	36	36	45
c-fat500-10.clq	500	46627	37.38	126	126	126	126	126
hamming6-4.clq	64	704	34.92	4	6	8	10	12
johnson8-2-4.clq	28	210	55.56	4	5	8	9	12
ln2-sanchis-100-40	100	1980	40.00	11	12	18	18	19
ln2-sanchis-100-50	100	2475	50.00	14	14	21	23	23
m200_-0.05	200	946	4.75	4	5	6	8	9
mulsol.i.1.col	197	3925	20.33	49	50	51	51	52
n5-sanchis-100-40	100	1980	40.00	20	20	20	20	21
n5-sanchis-200-40	200	7960	40.00	40	40	40	40	40
ln2-sanchis-100-40w	100	1980	40.00	568	763	928	1000	1073
ln2-sanchis-100-50w	100	2475	50.00	582	936	1115	1248	1318
m200-0	200	3916	19.68	14885	17893	22852	25824	30095
m300-0.01	300	9486	21.15	13962	18271	23417	26941	31538
m300-0	300	8779	19.57	13945	17361	22074	26594	29964
m400_-0.05	400	3611	4.53	9010	12590	15502	18612	22171
m500_-0.05	500	6122	4.91	9721	13031	18058	21170	23156
n5-sanchis-100-40w	100	1980	40.00	999	1132	1132	1132	1220
n5-sanchis-200-40w	200	7960	40.00	1810	2061	2061	2061	2061

Table 10 Number of k-plex verification routine calls

	Number of calls			
Graph	$k = 2$	$k = 3$	$k = 4$	$k = 5$
DSJR500.1.col	547598	6694248	105082633	920620676
MANN_a9.clq	58884	779585	95806974	15180
c-fat500-10.clq	676167	1282267	6498362	62778714
hamming6-4.clq	40330	392476	3676930	48342370
johnson8-2-4.clq	15042	69063	646741	938339
ln2-sanchis-100-40	1065358	6513307	187793367	3547760332
ln2-sanchis-100-50	5720933	8762473	348022549	10237520121
m200_-0.05	286888	9725368	93617693	1276171531
mulsol.i.1.col	120353	992520	23310124	184932820
n5-sanchis-100-40	92546	1681256	36210618	1048020088
n5-sanchis-200-40	415235	8200652	175073469	4157527829
ln2-sanchis-100-40w	231253	2130531	38423731	620325464
ln2-sanchis-100-50w	260628	2153812	45584585	661038617
m200-0	487722	8093854	122117459	1451143619
m300-0.01	1622679	38121918	746279875	11798946328
m300-0	1281898	23615834	437367455	6936706302
m400_-0.05	1342493	31687253	550408984	8348157174
m500_-0.05	3126802	110164439	3025629492	71922640064
n5-sanchis-100-40w	82048	1073733	10555688	144152407
n5-sanchis-200-40w	843975	16968442	386526747	8473020447

Table 11 Original and incremental k-plex verification routine

Graph	Runtime, sec							
	$k = 2$		$k = 3$		$k = 4$		$k = 5$	
	Original	Incremental	Original	Incremental	Original	Incremental	Original	Incremental
DSJR500.1.col	0.20	0.22	0.47	0.38	5.25	2.92	52.73	25.66
MANN_a9.clq	0.06	0.01	0.91	0.03	266.32	3.00	0.05	0.02
c-fat500-10.clq	2.44	0.24	2.95	0.22	9.30	0.30	89.56	1.52
hamming6-4.clq	0.02	0.02	0.08	0.05	0.78	0.16	12.91	1.75
johnson8-2-4.clq	0.01	0.01	0.03	0.01	0.30	0.05	0.55	0.05
ln2-sanchis-100-40	0.20	0.08	0.97	0.24	37.23	5.94	898.87	114.98
ln2-sanchis-100-50	1.52	0.22	1.83	0.31	106.38	11.38	3919.93	345.74
m200_-0.05	0.08	0.09	0.45	0.27	4.66	1.95	78.17	27.08
mulsol.i.1.col	0.09	0.08	0.14	0.09	1.38	0.48	11.84	3.45
n5-sanchis-100-40	0.05	0.05	0.28	0.09	6.77	1.13	262.33	34.06
n5-sanchis-200-40	0.16	0.09	2.98	0.27	76.17	4.33	1728.42	105.86
ln2-sanchis-100-40w	0.08	0.08	0.44	0.11	10.30	1.23	218.19	19.28
ln2-sanchis-100-50w	0.09	0.05	0.58	0.09	15.89	1.50	293.22	21.20
m200-0	0.11	0.09	0.78	0.25	13.56	2.52	188.72	28.20
m300-0.01	0.25	0.16	4.56	0.98	106.31	17.08	2007.31	261.14
m300-0	0.20	0.14	2.25	0.59	47.33	8.92	889.66	135.26
m400_-0.05	0.19	0.19	1.28	0.67	21.83	8.92	380.64	132.60
m500_-0.05	0.28	0.25	3.67	1.78	111.02	42.27	3042.08	984.60
n5-sanchis-100-40w	0.05	0.03	0.27	0.06	2.97	0.36	51.44	4.44
n5-sanchis-200-40w	0.22	0.09	3.70	0.53	116.99	10.55	3378.30	235.26

Table 12 5-plex in ln2-san-100-40w

Iteration	Best k-plex size		Best k-plex weight		Running time, sec	
	Smallest to largest	Larest to smallest	Smallest to largest	Larest to smallest	Smallest to largest	Larest to smallest
1	0	1	0	99	0.00	0.00
2	0	2	0	197	0.00	0.00
3	1	3	1	294	0.00	0.00
4	2	4	2	390	0.00	0.00
5	3	5	4	486	0.00	0.00
6	4	5	9	486	0.00	0.00
7	5	6	15	577	0.00	0.00
8	5	6	22	577	0.00	0.00
9	6	6	31	577	0.00	0.00
10	6	6	39	577	0.00	0.00
11	6	7	48	661	0.00	0.00
12	6	8	50	750	0.00	0.00
13	7	8	62	750	0.00	0.00
14	7	8	66	750	0.00	0.01
15	6	8	72	750	0.00	0.01
16	7	8	86	750	0.00	0.01
17	6	9	88	809	0.02	0.01
18	7	9	103	809	0.02	0.01
19	6	9	109	809	0.02	0.01
20	6	10	114	868	0.02	0.01
21	6	10	121	868	0.02	0.01
22	8	10	140	868	0.02	0.01
23	8	10	150	868	0.02	0.01
24	8	10	150	868	0.02	0.01
25	8	10	167	868	0.02	0.03
26	8	10	183	868	0.02	0.03
27	9	10	210	868	0.02	0.05
28	9	11	210	911	0.02	0.05
29	9	11	210	920	0.02	0.06
30	9	12	213	927	0.02	0.08

Table 12 (continued)

Iteration	Best k-plex size		Best k-plex weight		Running time, sec	
	Smallest to largest	Larest to smallest	Smallest to largest	Larest to smallest	Smallest to largest	Larest to smallest
31	9	12	213	927	0.02	0.11
32	9	12	216	927	0.02	0.14
33	9	12	222	967	0.02	0.20
34	9	12	222	967	0.02	0.23
35	10	12	253	967	0.03	0.28
36	10	13	263	985	0.03	0.36
37	10	13	277	985	0.03	0.42
38	10	13	277	985	0.05	0.48
39	11	13	295	985	0.05	0.63
40	10	13	296	985	0.05	0.88
41	11	13	324	985	0.06	1.05
42	11	14	327	1033	0.06	1.27
43	11	14	342	1033	0.08	1.58
44	11	14	360	1033	0.08	1.86
45	11	14	360	1033	0.09	2.69
46	12	14	376	1033	0.11	3.19
47	12	15	400	1083	0.11	3.67
48	12	15	410	1083	0.11	3.69
49	13	15	428	1083	0.11	3.73
50	13	15	428	1083	0.13	3.83
51	13	15	428	1083	0.14	4.22
52	13	15	447	1083	0.17	4.50
53	14	15	475	1083	0.19	4.88
54	14	15	475	1083	0.20	4.94
55	14	15	475	1083	0.22	5.73
56	14	15	475	1083	0.28	6.95
57	12	15	490	1083	0.31	7.69
58	12	16	513	1123	0.36	9.36
59	12	16	513	1123	0.41	9.67
60	12	17	523	1161	0.45	10.56

Table 12 (continued)

Iteration	Best k-plex size		Best k-plex weight		Running time, sec	
	Smallest to largest	Larest to smallest	Smallest to largest	Larest to smallest	Smallest to largest	Larest to smallest
61	12	17	558	1161	0.50	10.61
62	13	17	599	1161	0.53	11.11
63	13	17	599	1161	0.56	11.83
64	14	17	628	1161	0.63	12.72
65	14	18	628	1195	0.69	13.50
66	14	18	628	1195	0.75	13.55
67	14	18	628	1195	0.80	13.64
68	14	18	682	1195	0.94	13.94
69	15	18	743	1195	0.95	14.17
70	15	18	743	1195	0.97	14.38
71	15	18	743	1195	0.97	14.66
72	15	18	743	1195	1.02	15.88
73	15	18	743	1195	1.16	16.39
74	17	18	796	1195	1.30	17.48
75	15	18	802	1195	1.41	17.86
76	15	18	802	1195	1.53	22.75
77	15	18	843	1195	1.58	24.39
78	15	19	843	1220	1.67	26.34
79	15	19	843	1220	1.83	26.80
80	15	19	843	1220	1.92	27.22
81	16	19	847	1220	2.16	27.48
82	16	19	900	1220	2.38	28.27
83	16	19	900	1220	2.56	29.28
84	17	19	965	1220	2.70	29.97
85	17	19	965	1220	2.75	30.69
86	17	19	981	1220	2.86	31.14
87	16	19	987	1220	2.97	31.72
88	16	19	987	1220	3.27	32.44
89	17	19	1076	1220	3.64	33.50
90	17	19	1076	1220	3.78	34.42

Table 12 (continued)

Iteration	Best k-plex size		Best k-plex weight		Running time, sec	
	Smallest to largest	Larest to smallest	Smallest to largest	Larest to smallest	Smallest to largest	Larest to smallest
91	17	19	1076	1220	3.94	35.28
92	18	19	1110	1220	4.17	40.14
93	18	19	1110	1220	4.36	41.77
94	18	19	1110	1220	4.44	43.39
95	18	19	1110	1220	4.92	46.45
96	18	19	1110	1220	5.14	48.52
97	19	19	1161	1220	5.70	49.56
98	19	19	1161	1220	5.92	50.38
99	19	19	1220	1220	6.27	50.39
100	19	19	1220	1220	6.45	50.39

Table 13 Running time for weight based ordering

Graph	Running time, sec					
	$k = 2$			$k = 3$		
	No ordering	Ascending	Descending	No ordering	Ascending	Descending
ln2-sanchis-100-40w	0.06	0.05	0.06	0.31	0.09	0.66
ln2-sanchis-100-50w	0.08	0.05	0.13	0.69	0.11	0.41
m200-0	0.09	0.08	0.11	0.89	0.24	2.52
m300-0.01	0.31	0.16	0.55	8.38	1.00	31.55
m300-0	0.27	0.14	0.45	6.38	0.59	21.61
m400_-0.05	0.20	0.22	0.25	3.02	0.67	4.00
m500_-0.05	0.31	0.23	0.33	8.45	1.77	11.44
n5-sanchis-100-40w	0.06	0.03	0.03	0.13	0.08	0.06
n5-sanchis-200-40w	0.14	0.11	0.09	2.28	0.59	0.34
Graph	$k = 4$			$k = 5$		
	No ordering	Ascending	Descending	No ordering	Ascending	Descending
ln2-sanchis-100-40w	6.63	1.19	14.67	166.06	18.99	966.52
ln2-sanchis-100-50w	7.34	1.45	6.63	194.26	22.59	517.17
m200-0	18.67	2.52	110.30	380.54	29.08	> 3600
m300-0.01	465.12	17.14	1772.38	> 3600	261.37	> 3600
m300-0	242.14	8.95	1109.38	> 3600	136.71	> 3600
m400_-0.05	180.16	8.84	165.40	> 3600	131.05	> 3600
m500_-0.05	591.26	42.17	682.06	> 3600	984.55	> 3600
n5-sanchis-100-40w	3.98	0.38	1.72	89.29	4.55	33.86
n5-sanchis-200-40w	66.42	12.42	9.41	2506.74	261.01	> 3600

Table 14 Running time for Östergård ordering

Graph	Running time, sec					
	$k = 2$			$k = 3$		
	No ordering	Ascending	Descending	No ordering	Ascending	Descending
DSJR500.1.col	0.30	0.22	0.22	1.80	0.38	0.52
MANN_a9.clq	0.14	0.01	0.03	0.03	0.03	0.01
c-fat500-10.clq	0.20	0.24	0.30	0.25	0.22	0.22
hamming6-4.clq	0.03	0.02	0.02	0.03	0.05	0.03
johnson8-2-4.clq	0.01	0.01	0.03	0.02	0.01	0.01
ln2-sanchis-100-40	0.06	0.08	0.09	0.14	0.24	0.09
ln2-sanchis-100-50	0.25	0.22	0.70	0.19	0.31	0.16
m200_-0.05	0.08	0.09	0.13	0.66	0.27	0.24
mulsol.i.1.col	0.36	0.08	0.09	0.16	0.09	0.13
n5-sanchis-100-40	0.03	0.05	0.08	0.13	0.09	0.20
n5-sanchis-200-40	0.14	0.09	54.50	1.53	0.27	278.94
Graph	$k = 4$			$k = 5$		
	No ordering	Ascending	Descending	No ordering	Ascending	Descending
DSJR500.1.col	135.12	2.92	6.83	> 600	25.66	214.74
MANN_a9.clq	11.94	3.00	12.67	0.01	0.02	0.01
c-fat500-10.clq	0.78	0.30	0.44	8.41	1.52	2.50
hamming6-4.clq	0.16	0.16	0.16	1.75	1.75	1.74
johnson8-2-4.clq	0.06	0.05	0.05	0.09	0.05	0.09
ln2-sanchis-100-40	5.14	5.94	0.92	120.77	114.98	17.97
ln2-sanchis-100-50	5.88	11.38	2.03	264.05	345.74	43.75
m200_-0.05	20.31	1.95	4.51	625.78	27.08	84.63
mulsol.i.1.col	1.88	0.48	1.47	33.34	3.45	21.33
n5-sanchis-100-40	2.89	1.13	1.77	90.44	34.06	24.41
n5-sanchis-200-40	34.94	4.33	1048.32	933.05	105.86	> 600

Table 15 Running time for Östergård like ordering using double neighborhood

Graph	Running time, sec					
	$k = 2$			$k = 3$		
	No ordering	Ascending	Descending	No ordering	Ascending	Descending
DSJR500.1.col	0.19	0.19	0.22	0.19	0.19	1.17
MANN_a9.clq	0.05	0.05	0.11	0.22	0.22	0.03
c-fat500-10.clq	0.17	0.22	0.20	0.20	0.19	0.23
hamming6-4.clq	0.02	0.02	0.02	0.03	0.03	0.03
johnson8-2-4.clq	0.00	0.01	0.00	0.01	0.01	0.03
ln2-sanchis-100-40	0.08	0.08	0.08	0.19	0.19	0.14
ln2-sanchis-100-50	0.27	0.28	0.25	0.20	0.20	0.20
m200_-0.05	0.08	0.08	0.08	0.19	0.20	0.30
mulsol.i.1.col	0.13	0.06	0.08	0.16	0.08	0.13
n5-sanchis-100-40	0.05	0.05	0.03	0.16	0.17	0.11
n5-sanchis-200-40	0.14	0.14	0.16	2.02	2.00	1.50
Graph	$k = 4$			$k = 5$		
	No ordering	Ascending	Descending	No ordering	Ascending	Descending
DSJR500.1.col	0.20	0.19	41.55	0.25	0.17	-
MANN_a9.clq	14.30	14.31	11.98	0.03	0.02	0.02
c-fat500-10.clq	0.34	0.31	1.03	2.50	1.78	22.05
hamming6-4.clq	0.14	0.14	0.16	1.73	1.73	1.73
johnson8-2-4.clq	0.05	0.05	0.06	0.06	0.06	0.09
ln2-sanchis-100-40	4.20	4.22	5.14	145.69	145.87	120.94
ln2-sanchis-100-50	5.05	5.06	5.84	258.39	258.61	264.13
m200_-0.05	0.84	0.81	5.97	6.31	9.05	130.77
mulsol.i.1.col	0.25	0.13	1.39	0.73	0.42	14.70
n5-sanchis-100-40	3.23	3.25	2.91	138.55	138.48	89.91
n5-sanchis-200-40	44.19	44.13	34.80	1458.50	1457.47	929.86

Table 16 Running time for defective coloring

Graph	Running time, sec					
	No ordering	Defective m-coloring parameter				
		1	2	3	4	5
$k = 2$						
DSJR500.1.col	0.30	0.22	0.20	0.23	0.22	0.22
MANN_a9.clq	0.14	0.03	0.05	0.11	0.05	0.05
c-fat500-10.clq	0.20	0.20	0.19	0.20	0.19	0.19
hamming6-4.clq	0.03	0.03	0.02	0.02	0.02	0.05
johnson8-2-4.clq	0.01	0.01	0.01	0.01	0.01	0.00
ln2-sanchis-100-40	0.06	0.06	0.08	0.08	0.08	0.08
ln2-sanchis-100-50	0.25	0.61	0.27	0.28	0.39	0.33
m200_-0.05	0.08	0.08	0.08	0.09	0.08	0.08
mulsol.i.1.col	0.36	0.06	0.08	0.08	0.08	0.09
n5-sanchis-100-40	0.03	0.05	0.03	0.05	0.05	0.05
n5-sanchis-200-40	0.14	0.08	0.08	0.08	0.08	0.09
ln2-sanchis-100-40w	0.06	0.06	0.06	0.08	0.06	0.06
ln2-sanchis-100-50w	0.08	0.08	0.06	0.09	0.08	0.06
m200-0	0.09	0.11	0.11	0.09	0.09	0.11
m300-0.01	0.31	0.34	0.25	0.30	0.31	0.33
m300-0	0.27	0.22	0.25	0.30	0.30	0.25
m400_-0.05	0.20	0.19	0.20	0.20	0.20	0.20
m500_-0.05	0.31	0.25	0.30	0.27	0.30	0.30
n5-sanchis-100-40w	0.06	0.05	0.05	0.05	0.05	0.03
n5-sanchis-200-40w	0.14	0.11	0.16	0.09	0.11	0.11

Table 16 (continued)

Graph	Running time, sec					
	No ordering	Defective m-coloring parameter				
		1	2	3	4	5
$k = 3$						
DSJR500.1.col	1.80	0.80	0.86	1.08	1.44	1.48
MANN_a9.clq	0.03	0.34	0.98	0.02	0.66	0.95
c-fat500-10.clq	0.25	0.24	0.25	0.24	0.25	0.25
hamming6-4.clq	0.03	0.03	0.03	0.03	0.03	0.03
johnson8-2-4.clq	0.02	0.02	0.01	0.01	0.01	0.01
ln2-sanchis-100-40	0.14	0.17	0.13	0.13	0.09	0.11
ln2-sanchis-100-50	0.19	0.28	0.17	0.13	0.14	0.13
m200_-0.05	0.66	0.25	0.31	0.33	0.38	0.44
mulsol.i.1.col	0.16	0.08	0.19	0.13	0.14	0.13
n5-sanchis-100-40	0.13	0.09	0.08	0.09	0.06	0.08
n5-sanchis-200-40	1.53	0.33	0.16	0.17	0.19	0.19
ln2-sanchis-100-40w	0.31	0.28	0.25	0.30	0.20	0.20
ln2-sanchis-100-50w	0.69	0.41	0.23	0.45	0.45	0.38
m200-0	0.89	0.59	0.50	0.56	0.48	0.42
m300-0.01	8.38	7.06	4.24	6.19	6.66	8.92
m300-0	6.38	3.52	3.38	4.42	5.36	3.78
m400_-0.05	3.02	1.83	1.50	1.75	2.16	1.94
m500_-0.05	8.45	3.89	5.63	2.56	6.41	5.56
n5-sanchis-100-40w	0.13	0.19	0.09	0.13	0.09	0.13
n5-sanchis-200-40w	2.28	1.44	2.67	0.33	1.55	1.02

Table 16 (continued)

Graph	Running time, sec					
	No ordering	Defective m-coloring parameter				
		1	2	3	4	5
$k = 4$						
DSJR500.1.col	135.12	19.70	19.66	47.72	63.64	49.34
MANN_a9.clq	11.94	10.20	17.73	11.97	13.34	15.67
c-fat500-10.clq	0.78	0.63	0.66	0.78	0.84	0.88
hamming6-4.clq	0.16	0.16	0.14	0.16	0.16	0.14
johnson8-2-4.clq	0.06	0.03	0.05	0.05	0.05	0.05
ln2-sanchis-100-40	5.14	3.00	3.06	3.05	3.30	2.66
ln2-sanchis-100-50	5.88	21.84	6.59	4.20	4.38	4.25
m200_-0.05	20.31	3.42	4.42	5.26	13.06	13.86
mulsol.i.1.col	1.88	0.48	3.69	0.69	0.61	0.48
n5-sanchis-100-40	2.89	1.91	1.22	1.27	0.83	1.11
n5-sanchis-200-40	34.94	18.05	1.34	1.44	1.67	2.52
ln2-sanchis-100-40w	6.63	8.95	6.27	8.19	6.47	4.86
ln2-sanchis-100-50w	7.34	6.09	4.97	12.55	6.28	5.19
m200-0	18.67	12.91	12.47	17.05	11.64	8.64
m300-0.01	465.12	373.39	150.30	282.91	247.05	323.28
m300-0	242.14	115.14	105.41	178.88	187.58	119.21
m400_-0.05	180.16	74.36	68.98	94.02	107.68	91.35
m500_-0.05	591.26	173.63	237.79	180.19	305.76	258.45
n5-sanchis-100-40w	3.98	5.86	2.14	2.67	2.52	4.17
n5-sanchis-200-40w	66.42	55.88	75.07	10.56	29.67	36.81

Table 16 (continued)

Graph	Running time, sec					
	No ordering	Defective m-coloring parameter				
		1	2	3	4	5
$k = 5$						
DSJR500.1.col	>3599.32	793.33	921.40	2026.71	>3591.77	>3577.40
MANN_a9.clq	0.01	0.02	0.01	0.01	0.01	0.01
c-fat500-10.clq	8.41	6.08	6.42	8.45	9.36	9.27
hamming6-4.clq	1.75	1.74	1.75	1.75	1.75	1.75
johnson8-2-4.clq	0.09	0.05	0.06	0.06	0.11	0.08
ln2-sanchis-100-40	120.77	107.32	75.25	81.09	89.72	82.77
ln2-sanchis-100-50	264.05	725.25	253.98	141.26	218.08	286.39
m200_-0.05	625.78	80.21	137.41	118.24	280.32	326.55
mulsol.i.1.col	33.34	3.53	12.92	5.95	4.20	3.73
n5-sanchis-100-40	90.44	50.28	41.89	31.86	22.48	49.60
n5-sanchis-200-40	933.05	603.15	43.27	33.48	60.92	100.87
ln2-sanchis-100-40w	166.06	226.26	191.55	209.24	177.41	145.94
ln2-sanchis-100-50w	194.26	172.11	205.76	438.67	191.75	153.21
m200-0	380.54	290.39	285.80	486.09	232.82	182.37
m300-0.01	>3296.52	>3210.53	3274.64	>3422.35	>600.71	>3499.72
m300-0	>3444.73	3551.15	>2016.73	>3248.89	>3220.15	>2629.55
m400_-0.05	>3541.49	3096.88	2110.83	3366.76	3559.07	2590.41
m500_-0.05	>3571.75	>3519.03	>3598.01	>3571.89	>3548.22	>600.49
n5-sanchis-100-40w	89.29	126.77	70.58	60.25	85.10	124.47
n5-sanchis-200-40w	2506.74	3365.60	>3529.48	420.78	1183.04	2092.76

Table 17 Running time for defective coloring with reverse order

Graph	Running time, sec					
	No ordering	Defective m-coloring parameter				
		1	2	3	4	5
$k = 2$						
DSJR500.1.col	0.30	0.22	0.20	0.23	0.22	0.22
MANN_a9.clq	0.14	0.03	0.05	0.11	0.05	0.05
c-fat500-10.clq	0.20	0.20	0.19	0.20	0.19	0.19
hamming6-4.clq	0.03	0.03	0.02	0.02	0.02	0.05
johnson8-2-4.clq	0.01	0.01	0.01	0.01	0.01	0.00
ln2-sanchis-100-40	0.06	0.06	0.08	0.08	0.08	0.08
ln2-sanchis-100-50	0.25	0.61	0.27	0.28	0.39	0.33
m200_-0.05	0.08	0.08	0.08	0.09	0.08	0.08
mulsol.i.1.col	0.36	0.06	0.08	0.08	0.08	0.09
n5-sanchis-100-40	0.03	0.05	0.03	0.05	0.05	0.05
n5-sanchis-200-40	0.14	0.08	0.08	0.08	0.08	0.09
ln2-sanchis-100-40w	0.06	0.06	0.06	0.08	0.06	0.06
ln2-sanchis-100-50w	0.08	0.08	0.06	0.09	0.08	0.06
m200-0	0.09	0.11	0.11	0.09	0.09	0.11
m300-0.01	0.31	0.34	0.25	0.30	0.31	0.33
m300-0	0.27	0.22	0.25	0.30	0.30	0.25
m400_-0.05	0.20	0.19	0.20	0.20	0.20	0.20
m500_-0.05	0.31	0.25	0.30	0.27	0.30	0.30
n5-sanchis-100-40w	0.06	0.05	0.05	0.05	0.05	0.03
n5-sanchis-200-40w	0.14	0.11	0.16	0.09	0.11	0.11

Table 17 (continued)

Graph	Running time, sec					
	No ordering	Defective m-coloring parameter				
		1	2	3	4	5
$k = 3$						
DSJR500.1.col	1.80	0.80	0.86	1.08	1.44	1.48
MANN_a9.clq	0.03	0.34	0.98	0.02	0.66	0.95
c-fat500-10.clq	0.25	0.24	0.25	0.24	0.25	0.25
hamming6-4.clq	0.03	0.03	0.03	0.03	0.03	0.03
johnson8-2-4.clq	0.02	0.02	0.01	0.01	0.01	0.01
ln2-sanchis-100-40	0.14	0.17	0.13	0.13	0.09	0.11
ln2-sanchis-100-50	0.19	0.28	0.17	0.13	0.14	0.13
m200_-0.05	0.66	0.25	0.31	0.33	0.38	0.44
mulsol.i.1.col	0.16	0.08	0.19	0.13	0.14	0.13
n5-sanchis-100-40	0.13	0.09	0.08	0.09	0.06	0.08
n5-sanchis-200-40	1.53	0.33	0.16	0.17	0.19	0.19
ln2-sanchis-100-40w	0.31	0.28	0.25	0.30	0.20	0.20
ln2-sanchis-100-50w	0.69	0.41	0.23	0.45	0.45	0.38
m200-0	0.89	0.59	0.50	0.56	0.48	0.42
m300-0.01	8.38	7.06	4.24	6.19	6.66	8.92
m300-0	6.38	3.52	3.38	4.42	5.36	3.78
m400_-0.05	3.02	1.83	1.50	1.75	2.16	1.94
m500_-0.05	8.45	3.89	5.63	2.56	6.41	5.56
n5-sanchis-100-40w	0.13	0.19	0.09	0.13	0.09	0.13
n5-sanchis-200-40w	2.28	1.44	2.67	0.33	1.55	1.02

Table 17 (continued)

Graph	Running time, sec					
	No ordering	Defective m-coloring parameter				
		1	2	3	4	5
$k = 4$						
DSJR500.1.col	135.12	19.70	19.66	47.72	63.64	49.34
MANN_a9.clq	11.94	10.20	17.73	11.97	13.34	15.67
c-fat500-10.clq	0.78	0.63	0.66	0.78	0.84	0.88
hamming6-4.clq	0.16	0.16	0.14	0.16	0.16	0.14
johnson8-2-4.clq	0.06	0.03	0.05	0.05	0.05	0.05
ln2-sanchis-100-40	5.14	3.00	3.06	3.05	3.30	2.66
ln2-sanchis-100-50	5.88	21.84	6.59	4.20	4.38	4.25
m200_-0.05	20.31	3.42	4.42	5.26	13.06	13.86
mulsol.i.1.col	1.88	0.48	3.69	0.69	0.61	0.48
n5-sanchis-100-40	2.89	1.91	1.22	1.27	0.83	1.11
n5-sanchis-200-40	34.94	18.05	1.34	1.44	1.67	2.52
ln2-sanchis-100-40w	6.63	8.95	6.27	8.19	6.47	4.86
ln2-sanchis-100-50w	7.34	6.09	4.97	12.55	6.28	5.19
m200-0	18.67	12.91	12.47	17.05	11.64	8.64
m300-0.01	465.12	373.39	150.30	282.91	247.05	323.28
m300-0	242.14	115.14	105.41	178.88	187.58	119.21
m400_-0.05	180.16	74.36	68.98	94.02	107.68	91.35
m500_-0.05	591.26	173.63	237.79	180.19	305.76	258.45
n5-sanchis-100-40w	3.98	5.86	2.14	2.67	2.52	4.17
n5-sanchis-200-40w	66.42	55.88	75.07	10.56	29.67	36.81

Table 17 (continued)

Graph	Running time, sec					
	No ordering	Defective m-coloring parameter				
		1	2	3	4	5
$k = 5$						
DSJR500.1.col	>3599.32	793.33	921.40	2026.71	>3591.77	>3577.40
MANN_a9.clq	0.01	0.02	0.01	0.01	0.01	0.01
c-fat500-10.clq	8.41	6.08	6.42	8.45	9.36	9.27
hamming6-4.clq	1.75	1.74	1.75	1.75	1.75	1.75
johnson8-2-4.clq	0.09	0.05	0.06	0.06	0.11	0.08
ln2-sanchis-100-40	120.77	107.32	75.25	81.09	89.72	82.77
ln2-sanchis-100-50	264.05	725.25	253.98	141.26	218.08	286.39
m200_-0.05	625.78	80.21	137.41	118.24	280.32	326.55
mulsol.i.1.col	33.34	3.53	12.92	5.95	4.20	3.73
n5-sanchis-100-40	90.44	50.28	41.89	31.86	22.48	49.60
n5-sanchis-200-40	933.05	603.15	43.27	33.48	60.92	100.87
ln2-sanchis-100-40w	166.06	226.26	191.55	209.24	177.41	145.94
ln2-sanchis-100-50w	194.26	172.11	205.76	438.67	191.75	153.21
m200-0	380.54	290.39	285.80	486.09	232.82	182.37
m300-0.01	>3296.52	>3210.53	3274.64	>3422.35	>600.71	>3499.72
m300-0	>3444.73	3551.15	>2016.73	>3248.89	>3220.15	>2629.55
m400_-0.05	>3541.49	3096.88	2110.83	3366.76	3559.07	2590.41
m500_-0.05	>3571.75	>3519.03	>3598.01	>3571.89	>3548.22	>600.49
n5-sanchis-100-40w	89.29	126.77	70.58	60.25	85.10	124.47
n5-sanchis-200-40w	2506.74	3365.60	>3529.48	420.78	1183.04	2092.76

Table 18 Running time for weight based defective coloring

Graph	Running time, sec					
	No ordering	Defective m-coloring parameter				
		1	2	3	4	5
$k = 2$						
DSJR500.1.col	0.30	0.25	0.31	0.22	0.22	0.25
MANN_a9.clq	0.14	0.11	0.11	0.11	0.11	0.11
c-fat500-10.clq	0.20	0.20	0.20	0.20	0.19	0.22
hamming6-4.clq	0.03	0.03	0.03	0.01	0.03	0.03
johnson8-2-4.clq	0.01	0.01	0.01	0.01	0.02	0.01
ln2-sanchis-100-40	0.06	0.08	0.08	0.06	0.06	0.08
ln2-sanchis-100-50	0.25	0.25	0.25	0.25	0.25	0.25
m200_-0.05	0.08	0.09	0.09	0.09	0.08	0.08
mulsol.i.1.col	0.36	0.08	0.08	0.09	0.08	0.08
n5-sanchis-100-40	0.03	0.03	0.03	0.03	0.03	0.05
n5-sanchis-200-40	0.14	0.14	0.14	0.14	0.14	0.16
ln2-sanchis-100-40w	0.06	0.06	0.06	0.06	0.05	0.05
ln2-sanchis-100-50w	0.08	0.08	0.08	0.06	0.08	0.06
m200-0	0.09	0.11	0.09	0.09	0.09	0.09
m300-0.01	0.31	0.31	0.34	0.31	0.31	0.30
m300-0	0.27	0.27	0.28	0.27	0.28	0.27
m400_-0.05	0.20	0.20	0.20	0.20	0.20	0.20
m500_-0.05	0.31	0.31	0.31	0.31	0.31	0.31
n5-sanchis-100-40w	0.06	0.05	0.03	0.03	0.05	0.03
n5-sanchis-200-40w	0.14	0.16	0.14	0.16	0.14	0.14

Table 18 (continued)

Graph	Running time, sec					
	No ordering	Defective m-coloring parameter				
		1	2	3	4	5
$k = 3$						
DSJR500.1.col	1.80	1.80	1.81	1.81	1.81	1.81
MANN_a9.clq	0.03	0.02	0.03	0.03	0.03	0.03
c-fat500-10.clq	0.25	0.24	0.25	0.27	0.25	0.24
hamming6-4.clq	0.03	0.03	0.03	0.03	0.03	0.06
johnson8-2-4.clq	0.02	0.00	0.02	0.00	0.01	0.01
ln2-sanchis-100-40	0.14	0.16	0.14	0.16	0.14	0.16
ln2-sanchis-100-50	0.19	0.19	0.19	0.19	0.19	0.20
m200_-0.05	0.66	0.63	0.64	0.66	0.64	0.64
mulsol.i.1.col	0.16	0.14	0.13	0.13	0.14	0.13
n5-sanchis-100-40	0.13	0.11	0.11	0.14	0.11	0.11
n5-sanchis-200-40	1.53	1.50	1.52	1.53	1.52	1.52
ln2-sanchis-100-40w	0.31	0.33	0.33	0.31	0.31	0.31
ln2-sanchis-100-50w	0.69	0.69	0.69	0.69	0.69	0.67
m200-0	0.89	0.88	0.88	0.88	0.89	0.88
m300-0.01	8.38	8.34	8.36	8.39	8.34	8.36
m300-0	6.38	6.41	6.38	6.41	6.38	6.42
m400_-0.05	3.02	3.00	3.02	3.02	3.00	3.03
m500_-0.05	8.45	8.45	8.44	8.47	8.45	8.47
n5-sanchis-100-40w	0.13	0.11	0.13	0.11	0.13	0.13
n5-sanchis-200-40w	2.28	2.30	2.30	2.30	2.31	2.30

Table 18 (continued)

Graph	Running time, sec					
	No ordering	Defective m-coloring parameter				
		1	2	3	4	5
$k = 4$						
DSJR500.1.col	135.12	135.13	134.56	135.25	135.16	134.56
MANN_a9.clq	11.94	11.97	11.95	11.97	12.00	11.98
c-fat500-10.clq	0.78	0.78	0.80	0.77	0.78	0.78
hamming6-4.clq	0.16	0.16	0.16	0.16	0.17	0.16
johnson8-2-4.clq	0.06	0.05	0.05	0.05	0.05	0.05
ln2-sanchis-100-40	5.14	5.12	5.13	5.11	5.12	5.14
ln2-sanchis-100-50	5.88	5.83	5.84	5.84	5.84	5.86
m200_-0.05	20.31	20.33	20.41	20.38	20.36	20.45
mulsol.i.1.col	1.88	1.88	1.89	1.91	1.88	1.88
n5-sanchis-100-40	2.89	2.89	2.89	2.91	2.89	2.89
n5-sanchis-200-40	34.94	34.83	34.91	34.94	34.88	34.97
ln2-sanchis-100-40w	6.63	6.63	6.64	6.61	6.61	6.63
ln2-sanchis-100-50w	7.34	7.33	7.34	7.31	7.39	7.36
m200-0	18.67	18.64	18.63	18.56	18.63	18.64
m300-0.01	465.12	466.38	466.64	466.51	467.23	465.19
m300-0	242.14	242.16	242.56	241.22	242.05	243.38
m400_-0.05	180.16	179.51	180.31	180.19	179.48	180.11
m500_-0.05	591.26	589.18	591.24	589.79	592.27	591.16
n5-sanchis-100-40w	3.98	3.97	3.97	3.95	3.95	3.95
n5-sanchis-200-40w	66.42	66.31	66.15	66.33	66.09	66.33

Table 18 (continued)

Graph	Running time, sec					
	No ordering	Defective m-coloring parameter				
		1	2	3	4	5
$k = 5$						
DSJR500.1.col	>3599.32	>3539.71	>3593.49	>3595.96	>3539.63	>3596.29
MANN_a9.clq	0.01	0.01	0.01	0.02	0.01	0.03
c-fat500-10.clq	8.41	8.39	8.41	8.41	8.37	8.41
hamming6-4.clq	1.75	1.74	1.74	1.73	1.73	1.73
johnson8-2-4.clq	0.09	0.08	0.09	0.09	0.09	0.08
ln2-sanchis-100-40	120.77	120.85	121.35	120.95	120.95	120.92
ln2-sanchis-100-50	264.05	265.15	265.17	264.11	263.96	264.55
m200_-0.05	625.78	625.29	628.13	627.83	625.95	628.48
mulsol.i.1.col	33.34	33.39	33.28	33.44	33.47	33.42
n5-sanchis-100-40	90.44	90.11	89.86	90.19	90.22	90.25
n5-sanchis-200-40	933.05	929.33	933.61	933.06	934.25	934.36
ln2-sanchis-100-40w	166.06	166.33	165.61	166.10	166.27	165.68
ln2-sanchis-100-50w	194.26	195.02	194.19	194.88	194.97	194.23
m200-0	380.54	380.64	380.39	380.50	378.92	380.45
m300-0.01	>3296.52	>3296.87	>3312.96	>3299.03	>3312.00	>3310.36
m300-0	>3444.73	>3449.55	>3433.31	>3443.55	>3444.99	>3443.70
m400_-0.05	>3541.49	>3540.61	>3553.93	>848.63	>3552.81	>3554.38
m500_-0.05	>3571.75	>3571.21	>3558.34	>598.39	>3571.27	>3570.76
n5-sanchis-100-40w	89.29	90.19	89.64	89.28	89.58	89.66
n5-sanchis-200-40w	2506.74	2505.41	2513.45	>519.29	2515.88	2514.19

Table 19 Running time for weight based defective coloring, reverse order

Graph	Running time, sec					
	No ordering	Defective m-coloring parameter				
		1	2	3	4	5
$k = 2$						
DSJR500.1.col	0.30	0.25	0.31	0.22	0.22	0.25
MANN_a9.clq	0.14	0.11	0.11	0.11	0.11	0.11
c-fat500-10.clq	0.20	0.20	0.20	0.20	0.19	0.22
hamming6-4.clq	0.03	0.03	0.03	0.01	0.03	0.03
johnson8-2-4.clq	0.01	0.01	0.01	0.01	0.02	0.01
ln2-sanchis-100-40	0.06	0.08	0.08	0.06	0.06	0.08
ln2-sanchis-100-50	0.25	0.25	0.25	0.25	0.25	0.25
m200_-0.05	0.08	0.09	0.09	0.09	0.08	0.08
mulsol.i.1.col	0.36	0.08	0.08	0.09	0.08	0.08
n5-sanchis-100-40	0.03	0.03	0.03	0.03	0.03	0.05
n5-sanchis-200-40	0.14	0.14	0.14	0.14	0.14	0.16
ln2-sanchis-100-40w	0.06	0.06	0.06	0.06	0.05	0.05
ln2-sanchis-100-50w	0.08	0.08	0.08	0.06	0.08	0.06
m200-0	0.09	0.11	0.09	0.09	0.09	0.09
m300-0.01	0.31	0.31	0.34	0.31	0.31	0.30
m300-0	0.27	0.27	0.28	0.27	0.28	0.27
m400_-0.05	0.20	0.20	0.20	0.20	0.20	0.20
m500_-0.05	0.31	0.31	0.31	0.31	0.31	0.31
n5-sanchis-100-40w	0.06	0.05	0.03	0.03	0.05	0.03
n5-sanchis-200-40w	0.14	0.16	0.14	0.16	0.14	0.14

Table 19 (continued)

Graph	Running time, sec					
	No ordering	Defective m-coloring parameter				
		1	2	3	4	5
$k = 3$						
DSJR500.1.col	1.80	1.80	1.81	1.81	1.81	1.81
MANN_a9.clq	0.03	0.02	0.03	0.03	0.03	0.03
c-fat500-10.clq	0.25	0.24	0.25	0.27	0.25	0.24
hamming6-4.clq	0.03	0.03	0.03	0.03	0.03	0.06
johnson8-2-4.clq	0.02	0.00	0.02	0.00	0.01	0.01
ln2-sanchis-100-40	0.14	0.16	0.14	0.16	0.14	0.16
ln2-sanchis-100-50	0.19	0.19	0.19	0.19	0.19	0.20
m200_-0.05	0.66	0.63	0.64	0.66	0.64	0.64
mulsol.i.1.col	0.16	0.14	0.13	0.13	0.14	0.13
n5-sanchis-100-40	0.13	0.11	0.11	0.14	0.11	0.11
n5-sanchis-200-40	1.53	1.50	1.52	1.53	1.52	1.52
ln2-sanchis-100-40w	0.31	0.33	0.33	0.31	0.31	0.31
ln2-sanchis-100-50w	0.69	0.69	0.69	0.69	0.69	0.67
m200-0	0.89	0.88	0.88	0.88	0.89	0.88
m300-0.01	8.38	8.34	8.36	8.39	8.34	8.36
m300-0	6.38	6.41	6.38	6.41	6.38	6.42
m400_-0.05	3.02	3.00	3.02	3.02	3.00	3.03
m500_-0.05	8.45	8.45	8.44	8.47	8.45	8.47
n5-sanchis-100-40w	0.13	0.11	0.13	0.11	0.13	0.13
n5-sanchis-200-40w	2.28	2.30	2.30	2.30	2.31	2.30

Table 19 (continued)

Graph	Running time, sec					
	No ordering	Defective m-coloring parameter				
		1	2	3	4	5
$k = 4$						
DSJR500.1.col	135.12	135.13	134.56	135.25	135.16	134.56
MANN_a9.clq	11.94	11.97	11.95	11.97	12.00	11.98
c-fat500-10.clq	0.78	0.78	0.80	0.77	0.78	0.78
hamming6-4.clq	0.16	0.16	0.16	0.16	0.17	0.16
johnson8-2-4.clq	0.06	0.05	0.05	0.05	0.05	0.05
ln2-sanchis-100-40	5.14	5.12	5.13	5.11	5.12	5.14
ln2-sanchis-100-50	5.88	5.83	5.84	5.84	5.84	5.86
m200_-0.05	20.31	20.33	20.41	20.38	20.36	20.45
mulsol.i.1.col	1.88	1.88	1.89	1.91	1.88	1.88
n5-sanchis-100-40	2.89	2.89	2.89	2.91	2.89	2.89
n5-sanchis-200-40	34.94	34.83	34.91	34.94	34.88	34.97
ln2-sanchis-100-40w	6.63	6.63	6.64	6.61	6.61	6.63
ln2-sanchis-100-50w	7.34	7.33	7.34	7.31	7.39	7.36
m200-0	18.67	18.64	18.63	18.56	18.63	18.64
m300-0.01	465.12	466.38	466.64	466.51	467.23	465.19
m300-0	242.14	242.16	242.56	241.22	242.05	243.38
m400_-0.05	180.16	179.51	180.31	180.19	179.48	180.11
m500_-0.05	591.26	589.18	591.24	589.79	592.27	591.16
n5-sanchis-100-40w	3.98	3.97	3.97	3.95	3.95	3.95
n5-sanchis-200-40w	66.42	66.31	66.15	66.33	66.09	66.33

Table 19 (continued)

Graph	Running time, sec					
	No ordering	Defective m-coloring parameter				
		1	2	3	4	5
$k = 5$						
DSJR500.1.col	>3599.32	>3539.71	>3593.49	>3595.96	>3539.63	>3596.29
MANN_a9.clq	0.01	0.01	0.01	0.02	0.01	0.03
c-fat500-10.clq	8.41	8.39	8.41	8.41	8.37	8.41
hamming6-4.clq	1.75	1.74	1.74	1.73	1.73	1.73
johnson8-2-4.clq	0.09	0.08	0.09	0.09	0.09	0.08
ln2-sanchis-100-40	120.77	120.85	121.35	120.95	120.95	120.92
ln2-sanchis-100-50	264.05	265.15	265.17	264.11	263.96	264.55
m200_-0.05	625.78	625.29	628.13	627.83	625.95	628.48
mulsol.i.1.col	33.34	33.39	33.28	33.44	33.47	33.42
n5-sanchis-100-40	90.44	90.11	89.86	90.19	90.22	90.25
n5-sanchis-200-40	933.05	929.33	933.61	933.06	934.25	934.36
ln2-sanchis-100-40w	166.06	166.33	165.61	166.10	166.27	165.68
ln2-sanchis-100-50w	194.26	195.02	194.19	194.88	194.97	194.23
m200-0	380.54	380.64	380.39	380.50	378.92	380.45
m300-0.01	>3296.52	>3296.87	>3312.96	>3299.03	>3312.00	>3310.36
m300-0	>3444.73	>3449.55	>3433.31	>3443.55	>3444.99	>3443.70
m400_-0.05	>3541.49	>3540.61	>3553.93	>848.63	>3552.81	>3554.38
m500_-0.05	>3571.75	>3571.21	>3558.34	>598.39	>3571.27	>3570.76
n5-sanchis-100-40w	89.29	90.19	89.64	89.28	89.58	89.66
n5-sanchis-200-40w	2506.74	2505.41	2513.45	>519.29	2515.88	2514.19

Table 20 General and special algorithm for $k = 2$

Graph	Runtime, sec	
	General	Special
DSJR500.1.col	0.01	0.00
MANN_a9.clq	0.01	0.00
c-fat500-10.clq	0.03	0.00
hamming6-4.clq	0.00	0.00
johnson8-2-4.clq	0.00	0.00
ln2-sanchis-100-40	0.03	0.00
ln2-sanchis-100-50	0.14	0.00
m200_-0.05	0.01	0.00
mulsol.i.1.col	0.00	0.00
n5-sanchis-100-40	0.00	0.00
n5-sanchis-200-40	0.01	0.00
ln2-sanchis-100-40w	0.01	0.00
ln2-sanchis-100-50w	0.02	0.00
m200-0	0.02	0.00
m300-0.01	0.17	0.00
m300-0	0.03	0.00
m400_-0.05	0.03	0.00
m500_-0.05	0.05	0.00
n5-sanchis-100-40w	0.00	0.00
n5-sanchis-200-40w	0.05	0.00

Table 21 Running time with N_2 based pruning

Graph	Runtime, sec							
	$k = 2$		$k = 3$		$k = 4$		$k = 5$	
	Without pruning	With pruning	Without pruning	With pruning	Without pruning	With pruning	Without pruning	With pruning
DSJR500.1.col	0.28	0.01	4.38	0.14	256.05	1.94	> 600	17.28
MANN_a9.clq	0.03	0.01	0.05	0.02	2.81	2.20	0.01	0.00
c-fat500-10.clq	0.22	0.03	0.25	0.03	0.45	0.09	3.14	0.98
hamming6-4.clq	0.03	0.00	0.05	0.00	0.14	0.08	1.55	1.25
johnson8-2-4.clq	0.01	0.00	0.01	0.00	0.05	0.03	0.05	0.03
ln2-sanchis-100-40	0.08	0.03	0.22	0.14	5.22	4.14	100.91	81.28
ln2-sanchis-100-50	0.20	0.14	0.30	0.19	10.19	8.30	308.00	255.37
m200_-0.05	0.13	0.01	0.83	0.14	17.91	1.27	822.45	18.11
mulsol.i.1.col	0.08	0.00	0.20	0.02	5.58	0.30	102.77	2.23
n5-sanchis-100-40	0.05	0.00	0.09	0.03	1.00	0.78	29.98	24.20
n5-sanchis-200-40	0.08	0.01	0.27	0.13	3.77	3.00	92.44	75.34
ln2-sanchis-100-40w	0.05	0.01	0.09	0.06	1.08	0.91	17.14	14.56
ln2-sanchis-100-50w	0.05	0.02	0.09	0.06	1.34	1.11	18.98	15.72
m200-0	0.08	0.02	0.20	0.16	2.13	1.80	23.56	19.72
m300-0.01	0.13	0.17	0.86	0.72	14.56	11.89	220.76	187.10
m300-0	0.13	0.03	0.52	0.38	7.45	6.03	112.13	94.28
m400_-0.05	0.17	0.03	0.98	0.38	24.89	5.86	751.20	88.44
m500_-0.05	0.20	0.05	2.03	1.11	70.72	27.98	2521.03	654.25
n5-sanchis-100-40w	0.03	0.00	0.06	0.03	0.31	0.23	3.98	3.25
n5-sanchis-200-40w	0.08	0.05	0.47	0.34	9.36	7.56	207.91	175.99

APPENDIX B

EXACT k-PLEX ALGORITHM NUMERICAL RESULTS AND COMPARISONS

Table 22 Dimacs and Sanchis instances information

Graph	Nodes	Edges	Density, %	Maximum k-plex size				
				$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
C125.9.clq	125	6963	89.85	34	43	≥35	≥48	≥51
C250.9.clq	250	27984	89.91	≥40	≥31	≥31	≥51	≥54
C500.9.clq	500	112332	90.05	≥41	≥30	≥28	≥28	≥51
DSJC125.1.col	125	736	9.50	4	5	7	8	9
DSJC125.5.col	125	3891	50.21	10	13	14	17	≥18
DSJC125.9.col	125	6961	89.82	34	≥40	≥35	≥48	≥50
DSJC250.1.col	250	3218	10.34	4	6	7	8	10
DSJC250.5.col	250	15668	50.34	12	14	17	≥16	≥17
DSJC250.9.col	250	27897	89.63	≥38	≥31	≥42	≥47	≥50
DSJC500.1.col	500	12458	9.99	5	6	8	9	≥10
DSJC500.5.clq	500	62624	50.20	13	16	≥15	≥15	≥15
DSJC500.5.col	500	62624	50.20	13	16	≥15	≥15	≥15
DSJC500.9.col	500	112437	90.13	≥40	≥29	≥29	≥43	≥55
DSJR500.1.col	500	3555	2.85	12	14	15	15	16
DSJR500.1c.col	500	121275	97.21	≥35	≥37	≥54	≥95	≥95
DSJR500.5.col	500	58862	47.18	≥120	123	≥115	≥89	≥84
MANN_a27.clq	378	70551	99.01	126	≥235	351	≥350	≥350
MANN_a9.clq	45	918	92.73	16	26	36	36	44
R125.1.col	125	209	2.70	5	6	6	7	9
R125.1c.col	125	7501	96.79	46	70	87	100	109
R125.5.col	125	3838	49.52	36	36	38	39	40
R250.1.col	250	867	2.79	8	8	9	11	11
R250.1c.col	250	30227	97.11	≥49	≥54	≥63	≥90	≥122
R250.5.col	250	14849	47.71	65	65	67	68	≥60
brock200_1.clq	200	14834	74.54	21	≥24	≥23	≥23	≥23
brock200_2.clq	200	9876	49.63	12	13	16	≥17	≥16

Table 22 (continued)

Graph	Nodes	Edges	Density, %	Maximum k-plex size				
				$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
brock200_3.clq	200	12048	60.54	15	17	≥19	≥19	≥19
brock200_4.clq	200	13089	65.77	17	20	≥20	≥20	≥19
brock400_1.clq	400	59723	74.84	27	≥23	≥22	≥22	≥21
brock400_2.clq	400	59786	74.92	29	≥23	≥22	≥21	≥21
brock400_3.clq	400	59681	74.79	31	≥22	≥22	≥21	≥22
brock400_4.clq	400	59765	74.89	33	≥22	≥22	≥21	≥22
c-fat200-1.clq	200	1534	7.71	12	12	12	12	14
c-fat200-2.clq	200	3235	16.26	24	24	24	24	24
c-fat200-5.clq	200	8473	42.58	58	58	58	58	58
c-fat500-1.clq	500	4459	3.57	14	14	14	14	15
c-fat500-10.clq	500	46627	37.38	126	126	126	126	126
c-fat500-2.clq	500	9139	7.33	26	26	26	26	26
c-fat500-5.clq	500	23191	18.59	64	64	64	64	64
flat300_20_0.col	300	21375	47.66	11	14	17	≥16	≥17
flat300_26_0.col	300	21633	48.23	11	14	≥16	≥16	≥16
flat300_28_0.col	300	21695	48.37	12	14	≥16	≥16	≥16
gen200_p0.9_44.clq	200	17910	90.00	44	≥33	≥41	≥50	≥55
gen200_p0.9_55.clq	200	17910	90.00	55	≥49	≥33	≥53	≥57
gen400_p0.9_55.clq	400	71820	90.00	55	≥39	≥29	≥53	≥61
gen400_p0.9_65.clq	400	71820	90.00	65	≥30	≥28	≥28	≥54
gen400_p0.9_75.clq	400	71820	90.00	75	≥54	≥40	≥43	≥60
hamming6-2.clq	64	1824	90.48	32	32	32	40	48
hamming6-4.clq	64	704	34.92	4	6	8	10	12
hamming8-2.clq	256	31616	96.86	128	128	128	≥45	≥60
hamming8-4.clq	256	20864	63.92	16	16	≥20	≥18	≥18
johnson16-2-4.clq	120	5460	76.47	8	10	≥16	≥19	≥21
johnson32-2-4.clq	496	107880	87.88	16	≥21	≥24	≥25	≥26
johnson8-2-4.clq	28	210	55.56	4	5	8	9	12
johnson8-4-4.clq	70	1855	76.81	14	14	18	22	≥24
keller4.clq	171	9435	64.91	11	15	21	≥22	≥20
le450_15a.col	450	8168	8.09	15	15	15	15	15
le450_15b.col	450	8169	8.09	15	15	15	15	15
le450_15c.col	450	16680	16.51	15	15	15	16	≥15

Table 22 (continued)

Graph	Nodes	Edges	Density, %	Maximum k-plex size				
				$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
le450_15d.col	450	16750	16.58	15	15	15	15	≥14
ln2-sanchis-100-40	100	1980	40.00	11	12	18	18	19
ln2-sanchis-100-50	100	2475	50.00	14	14	21	23	23
ln2-sanchis-100-60	100	2970	60.00	19	20	23	24	26
ln2-sanchis-100-70	100	3465	70.00	26	26	26	≥26	≥26
ln2-sanchis-100-80	100	3960	80.00	42	42	42	42	≥40
ln2-sanchis-100-90	100	4455	90.00	88	88	88	88	88
ln2-sanchis-1000-40	1000	199800	40.00	16	≥16	≥11	≥30	≥33
ln2-sanchis-1000-50	1000	249750	50.00	20	≥20	≥13	≥13	≥38
ln2-sanchis-1000-60	1000	299700	60.00	28	≥16	≥16	≥46	≥55
ln2-sanchis-1000-70	1000	349650	70.00	39	≥18	≥18	≥74	≥94
ln2-sanchis-1000-80	1000	399600	80.00	62	≥21	≥86	≥22	≥31
ln2-sanchis-1000-90	1000	449550	90.00	132	≥89	≥198	≥59	≥60
ln2-sanchis-200-40	200	7960	40.00	12	13	≥19	≥24	≥30
ln2-sanchis-200-50	200	9950	50.00	16	17	≥24	≥32	≥36
ln2-sanchis-200-60	200	11940	60.00	21	23	33	≥37	≥38
ln2-sanchis-200-70	200	13930	70.00	30	≥31	45	≥39	≥21
ln2-sanchis-200-80	200	15920	80.00	48	48	≥48	≥24	≥24
ln2-sanchis-200-90	200	17910	90.00	101	101	101	101	≥58
ln2-sanchis-300-40	300	17940	40.00	13	≥15	≥21	≥28	≥32
ln2-sanchis-300-50	300	22425	50.00	17	≥18	≥24	≥36	≥33
ln2-sanchis-300-60	300	26910	60.00	23	≥23	≥33	≥40	≥43
ln2-sanchis-300-70	300	31395	70.00	32	≥32	≥43	≥53	≥18
ln2-sanchis-300-80	300	35880	80.00	52	≥52	≥62	≥23	≥33
ln2-sanchis-300-90	300	40365	90.00	109	≥99	≥93	≥56	≥62
ln2-sanchis-400-40	400	31920	40.00	14	≥15	≥21	≥28	≥30
ln2-sanchis-400-50	400	39900	50.00	18	≥19	≥27	≥33	≥43
ln2-sanchis-400-60	400	47880	60.00	24	≥24	≥33	≥36	≥60
ln2-sanchis-400-70	400	55860	70.00	34	≥34	≥32	≥59	≥19
ln2-sanchis-400-80	400	63840	80.00	54	≥47	≥79	≥81	≥22
ln2-sanchis-400-90	400	71820	90.00	114	≥88	≥138	≥132	≥66
ln2-sanchis-500-40	500	49900	40.00	14	≥15	≥21	≥25	≥23
ln2-sanchis-500-50	500	62375	50.00	18	≥19	≥27	≥34	≥43

Table 22 (continued)

Graph	Nodes	Edges	Density, %	Maximum k-plex size				
				$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
ln2-sanchis-500-60	500	74850	60.00	25	≥25	≥16	≥42	≥65
ln2-sanchis-500-70	500	87325	70.00	35	≥35	≥18	≥72	≥18
ln2-sanchis-500-80	500	99800	80.00	56	≥56	≥84	≥21	≥22
ln2-sanchis-500-90	500	112275	90.00	118	≥95	≥28	≥61	≥68
ln2-sanchis-600-40	600	71880	40.00	14	≥15	≥21	≥27	≥32
ln2-sanchis-600-50	600	89850	50.00	19	≥19	≥29	≥39	≥38
ln2-sanchis-600-60	600	107820	60.00	26	≥26	≥35	≥42	≥63
ln2-sanchis-600-70	600	125790	70.00	36	≥18	≥18	≥71	≥30
ln2-sanchis-600-80	600	143760	80.00	58	≥22	≥82	≥23	≥29
ln2-sanchis-600-90	600	161730	90.00	122	≥115	≥27	≥51	≥53
ln2-sanchis-700-40	700	97860	40.00	15	≥15	≥18	≥26	≥34
ln2-sanchis-700-50	700	122325	50.00	19	≥19	≥30	≥32	≥32
ln2-sanchis-700-60	700	146790	60.00	26	≥26	≥15	≥39	≥60
ln2-sanchis-700-70	700	171255	70.00	37	≥19	≥18	≥74	≥18
ln2-sanchis-700-80	700	195720	80.00	59	≥22	≥87	≥21	≥21
ln2-sanchis-700-90	700	220185	90.00	125	≥124	≥122	≥59	≥61
ln2-sanchis-800-40	800	127840	40.00	15	≥15	≥21	≥22	≥20
ln2-sanchis-800-50	800	159800	50.00	20	≥20	≥13	≥24	≥38
ln2-sanchis-800-60	800	191760	60.00	27	≥15	≥15	≥45	≥64
ln2-sanchis-800-70	800	223720	70.00	38	≥18	≥37	≥76	≥43
ln2-sanchis-800-80	800	255680	80.00	60	≥22	≥84	≥21	≥31
ln2-sanchis-800-90	800	287640	90.00	127	≥96	≥26	≥61	≥69
ln2-sanchis-900-40	900	161820	40.00	15	≥15	≥16	≥26	≥24
ln2-sanchis-900-50	900	202275	50.00	20	≥21	≥14	≥26	≥31
ln2-sanchis-900-60	900	242730	60.00	27	≥26	≥37	≥46	≥41
ln2-sanchis-900-70	900	283185	70.00	39	≥18	≥17	≥75	≥28
ln2-sanchis-900-80	900	323640	80.00	61	≥21	≥22	≥21	≥21
ln2-sanchis-900-90	900	364095	90.00	130	≥27	≥26	≥57	≥63
mulsol.i.1.col	197	3925	20.33	49	50	51	51	52
n5-sanchis-100-40	100	1980	40.00	20	20	20	20	21
n5-sanchis-100-50	100	2475	50.00	20	20	20	21	22
n5-sanchis-100-60	100	2970	60.00	20	20	22	24	26
n5-sanchis-100-70	100	3465	70.00	20	20	24	27	≥25

Table 22 (continued)

Graph	Nodes	Edges	Density, %	Maximum k-plex size				
				$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
n5-sanchis-100-80	100	3960	80.00	20	24	29	≥28	≥33
n5-sanchis-100-90	100	4455	90.00	20	37	≥41	≥47	≥53
n5-sanchis-1000-40	1000	199800	40.00	200	200	200	≥200	≥200
n5-sanchis-1000-50	1000	249750	50.00	200	200	200	≥200	≥200
n5-sanchis-1000-60	1000	299700	60.00	200	200	≥200	≥200	≥128
n5-sanchis-1000-70	1000	349650	70.00	200	≥148	≥200	≥200	≥19
n5-sanchis-1000-80	1000	399600	80.00	200	≥200	≥120	≥21	≥22
n5-sanchis-1000-90	1000	449550	90.00	200	≥29	≥25	≥57	≥60
n5-sanchis-200-40	200	7960	40.00	40	40	40	40	40
n5-sanchis-200-50	200	9950	50.00	40	40	40	40	≥40
n5-sanchis-200-60	200	11940	60.00	40	40	40	≥40	≥40
n5-sanchis-200-70	200	13930	70.00	40	40	≥40	≥40	≥20
n5-sanchis-200-80	200	15920	80.00	40	≥40	≥40	≥25	≥25
n5-sanchis-200-90	200	17910	90.00	40	≥31	≥30	≥31	≥54
n5-sanchis-300-40	300	17940	40.00	60	60	60	60	≥60
n5-sanchis-300-50	300	22425	50.00	60	60	60	60	≥60
n5-sanchis-300-60	300	26910	60.00	60	60	60	≥60	≥38
n5-sanchis-300-70	300	31395	70.00	60	60	≥60	≥60	≥19
n5-sanchis-300-80	300	35880	80.00	60	≥60	≥30	≥24	≥29
n5-sanchis-300-90	300	40365	90.00	60	≥30	≥30	≥57	≥62
n5-sanchis-400-40	400	31920	40.00	80	80	80	80	≥80
n5-sanchis-400-50	400	39900	50.00	80	80	80	80	≥80
n5-sanchis-400-60	400	47880	60.00	80	80	80	≥80	≥61
n5-sanchis-400-70	400	55860	70.00	80	80	≥80	≥56	≥19
n5-sanchis-400-80	400	63840	80.00	80	≥80	≥80	≥22	≥23
n5-sanchis-400-90	400	71820	90.00	80	≥32	≥28	≥56	≥60
n5-sanchis-500-40	500	49900	40.00	100	100	100	100	≥100
n5-sanchis-500-50	500	62375	50.00	100	100	100	≥100	≥100
n5-sanchis-500-60	500	74850	60.00	100	100	100	≥100	≥80
n5-sanchis-500-70	500	87325	70.00	100	100	≥100	≥100	≥19
n5-sanchis-500-80	500	99800	80.00	100	≥100	≥100	≥23	≥23
n5-sanchis-500-90	500	112275	90.00	100	≥31	≥27	≥56	≥63
n5-sanchis-600-40	600	71880	40.00	120	120	120	120	≥120

Table 22 (continued)

Graph	Nodes	Edges	Density, %	Maximum k-plex size				
				$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
n5-sanchis-600-50	600	89850	50.00	120	120	120	≥120	≥120
n5-sanchis-600-60	600	107820	60.00	120	120	120	≥120	≥120
n5-sanchis-600-70	600	125790	70.00	120	120	≥120	≥86	≥19
n5-sanchis-600-80	600	143760	80.00	120	≥120	≥120	≥23	≥22
n5-sanchis-600-90	600	161730	90.00	120	≥28	≥28	≥57	≥67
n5-sanchis-700-40	700	97860	40.00	140	140	140	140	≥140
n5-sanchis-700-50	700	122325	50.00	140	140	140	≥140	≥140
n5-sanchis-700-60	700	146790	60.00	140	140	≥140	≥140	≥39
n5-sanchis-700-70	700	171255	70.00	140	140	≥140	≥97	≥18
n5-sanchis-700-80	700	195720	80.00	140	≥140	≥22	≥21	≥21
n5-sanchis-700-90	700	220185	90.00	140	≥27	≥28	≥50	≥74
n5-sanchis-800-40	800	127840	40.00	160	160	160	≥152	≥160
n5-sanchis-800-50	800	159800	50.00	160	160	160	≥160	≥160
n5-sanchis-800-60	800	191760	60.00	160	160	≥131	≥160	≥108
n5-sanchis-800-70	800	223720	70.00	160	160	≥160	≥160	≥18
n5-sanchis-800-80	800	255680	80.00	160	≥160	≥22	≥21	≥29
n5-sanchis-800-90	800	287640	90.00	160	≥28	≥29	≥58	≥65
n5-sanchis-900-40	900	161820	40.00	180	180	180	≥180	≥180
n5-sanchis-900-50	900	202275	50.00	180	180	180	≥180	≥180
n5-sanchis-900-60	900	242730	60.00	180	180	≥154	≥180	≥92
n5-sanchis-900-70	900	283185	70.00	180	180	≥180	≥73	≥18
n5-sanchis-900-80	900	323640	80.00	180	≥180	≥180	≥21	≥21
n5-sanchis-900-90	900	364095	90.00	180	≥27	≥27	≥60	≥66
p_hat300-1.clq	300	10933	24.38	8	10	12	14	≥13
p_hat300-2.clq	300	21928	48.89	25	30	≥30	≥19	≥18
p_hat300-3.clq	300	33390	74.45	36	≥26	≥22	≥20	≥20
p_hat500-1.clq	500	31569	25.31	9	12	14	≥13	≥14
p_hat500-2.clq	500	62946	50.46	36	≥23	≥19	≥16	≥15
p_hat500-3.clq	500	93800	75.19	≥49	≥22	≥19	≥18	≥69
r100.5	100	2508	50.67	9	12	14	16	18
r200.5	200	10036	50.43	11	14	17	≥17	≥17
r300.5	300	22361	49.86	12	14	≥17	≥16	≥16
r400.5	400	40061	50.20	13	15	≥16	≥16	≥16

Table 22 (continued)

Graph	Nodes	Edges	Density, %	Maximum k-plex size				
				$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
r500.5	500	62161	49.83	13	16	≥16	≥15	≥16
san200_0.7_1.clq	200	13930	70.00	30	≥30	≥43	≥57	≥72
san200_0.7_2.clq	200	13930	70.00	18	≥24	≥31	≥44	≥51
san200_0.9_1.clq	200	17910	90.00	70	≥82	124	125	≥124
san200_0.9_2.clq	200	17910	90.00	60	≥59	105	≥79	≥54
san200_0.9_3.clq	200	17910	90.00	44	≥45	≥58	≥73	≥43
san400_0.5_1.clq	400	39900	50.00	13	≥11	≥21	≥28	≥34
san400_0.7_1.clq	400	55860	70.00	40	≥27	≥45	≥57	≥89
san400_0.7_2.clq	400	55860	70.00	30	≥24	≥41	≥55	≥65
san400_0.7_3.clq	400	55860	70.00	22	≥17	≥17	≥45	≥56
san400_0.9_1.clq	400	71820	90.00	100	≥81	≥113	≥153	≥109
sanr200_0.7.clq	200	13868	69.69	18	22	≥21	≥21	≥21
sanr200_0.9.clq	200	17863	89.76	42	≥34	≥33	≥33	≥49
sanr400_0.5.clq	400	39984	50.11	13	15	≥15	≥16	≥16
sanr400_0.7.clq	400	55869	70.01	21	≥20	≥20	≥20	≥20
school1.col	385	19095	25.83	14	≥28	≥38	≥40	≥41
school1_nsh.col	352	14612	23.65	14	28	37	41	≥42

Table 23 Dimacs and Sanchis instances running time

Graph	$k=2$		$k=3$		$k=4$		$k=5$	
	Ordering	Running time, sec	Ordering	Running time, sec	Ordering	Running time, sec	Ordering	Running time, sec
C125.9.clq	O1	6186.06	O1R	>10800	O1	>10800	O1	>10800
C250.9.clq	O1R	>10800	O1R	>10800	O1	>10800	O1	>10800
C500.9.clq	O1R	>10800	O1R	>10800	O1R	>10800	O1	>10800
DSJC125.1.col	O1R	0.06	W4R	0.08	O2	0.39	C1	13.25
DSJC125.5.col	O1R	0.28	O1R	19.17	O1R	1239.17	O1R	>10800
DSJC125.9.col	O1	>10800	O1R	>10800	O1	>10800	O1	>10800
DSJC250.1.col	O1R	0.13	O1R	1.13	O1R	58.97	O1R	1489.09
DSJC250.5.col	O1R	23.98	O1R	4547.94	O1R	>10800	O1R	>10800
DSJC250.9.col	O1R	>10800	O1	>10800	O1	>10800	O1	>10800
DSJC500.1.col	O1R	0.47	O1R	18.44	C2R	4576.47	W4	>10800
DSJC500.5.clq	O1R	6457.14	O1R	>10800	O1R	>10800	O1R	>10800
DSJC500.5.col	O1R	6465.76	O1R	>10800	O1R	>10800	O1R	>10800
DSJC500.9.col	O1R	>10800	O1R	>10800	W4R	>10800	C4R	>10800
DSJR500.1.col	D	0.20	D	0.19	O2	0.22	O2	0.45
DSJR500.1c.col	C1	>10800	C1	>10800	C2R	>10800	C3R	>10800
DSJR500.5.col	C1	1030.65	C1	>10800	C1	>10800	O1R	>10800
MANN_a27.clq	D	>10800	O1R	0.24	W2R	>10800	O2	>10800
MANN_a9.clq	O1R	0.01	O1R	0.03	C1	6.23	DR	0.03
R125.1.col	D	0.05	W2R	0.03	C1R	0.06	W3R	0.06
R125.1c.col	O1R	15.30	O1R	4465.74	O1	1.44	O1R	0.19
R125.5.col	DR	0.08	C1	0.30	C1	10.11	C3R	53.49
R250.1.col	O1	0.09	C1R	0.09	D	0.09	C4R	0.09
R250.1c.col	O1	>10800	O1	>10800	O1	>10800	O1	>10800
R250.5.col	C2R	0.59	C5R	15.27	C3R	154.19	C1	>10800
brock200_1.clq	O1R	>10800	O1R	>10800	O1R	>10800	O1R	>10800
brock200_2.clq	O1	4.74	O1R	446.68	O1R	>10800	O1R	>10800
brock200_3.clq	O1R	76.33	O1R	>10800	O1R	>10800	O1R	>10800
brock200_4.clq	O1R	398.65	O1R	>10800	O1R	>10800	O1R	>10800
brock400_1.clq	O1R	>10800	O1R	>10800	O1R	>10800	O1R	>10800
brock400_2.clq	O1R	>10800	O1R	>10800	O1R	>10800	O1R	>10800

Table 23 (continued)

Graph	$k = 2$		$k = 3$		$k = 4$		$k = 5$	
	Ordering	Running time, sec	Ordering	Running time, sec	Ordering	Running time, sec	Ordering	Running time, sec
brock400_3.clq	O1R	>10800	O1R	>10800	O1R	>10800	O1R	>10800
brock400_4.clq	O1R	>10800	O1R	>10800	O1R	>10800	O1R	>10800
c-fat200-1.clq	O1R	0.06	C5R	0.08	W1R	0.08	D	0.09
c-fat200-2.clq	O1	0.08	O1	0.08	O1	0.11	O1	0.20
c-fat200-5.clq	D	0.08	D	0.08	C4R	0.13	O2	0.75
c-fat500-1.clq	C5R	0.16	C5R	0.17	O2	0.27	O2	0.24
c-fat500-10.clq	O1	0.19	O1	0.22	O2	0.42	O1	1.70
c-fat500-2.clq	O1R	0.19	C5R	0.17	D	0.31	O2	1.41
c-fat500-5.clq	O1	0.17	O1	0.19	O2	0.30	O1	1.03
flat300_20_0.col	O1R	27.36	C5	10454.7	O1R	>10800	C1	>10800
flat300_26_0.col	D	47.08	O1R	>10800	O1R	>10800	O1R	>10800
flat300_28_0.col	C1	53.13	O1R	>10800	O1R	>10800	O1R	>10800
gen200_p0.9_44.clq	O1R	>10800	C1	>10800	O1	>10800	O1	>10800
gen200_p0.9_55.clq	O1R	>10800	O1R	>10800	O1	>10800	O1	>10800
gen400_p0.9_55.clq	C1	>10800	O1R	>10800	O1	>10800	O1	>10800
gen400_p0.9_65.clq	O1R	>10800	O1R	>10800	O1R	>10800	O1	>10800
gen400_p0.9_75.clq	O1R	>10800	C1	>10800	C4	>10800	O1	>10800
hamming6-2.clq	O1R	0.03	O1R	0.14	W5	131.86	DR	16.11
hamming6-4.clq	D	0.02	O1R	0.03	O1R	0.16	O1R	1.75
hamming8-2.clq	W3R	0.13	C3R	1500.83	C4R	>10800	W5	>10800
hamming8-4.clq	C5	7.45	W4	>10800	C4	>10800	O2R	>10800
johnson16-2-4.clq	C1R	3124.10	C4	>10800	C5	>10800	O2	>10800
johnson32-2-4.clq	W1R	>10800	D	>10800	D	>10800	O1	>10800
johnson8-2-4.clq	O1	0.01	O1R	0.01	O2	0.05	O1	0.06
johnson8-4-4.clq	DR	0.05	W3R	4.92	O2	769.75	C4R	>10800
keller4.clq	W5	55.45	O2	5356.47	O1R	>10800	O1R	>10800
le450_15a.col	C1	0.19	O1R	1.48	O1R	56.72	O2R	8253.35
le450_15b.col	C1	0.19	C1	1.20	O1R	65.55	O2R	9804.87
le450_15c.col	O1	0.28	C4R	10.11	C1	894.74	C1	>10800
le450_15d.col	O1	0.30	C3R	11.39	O1R	1399.44	O1R	>10800
ln2-sanchis-100-40	D	0.08	O1R	0.09	O1R	0.94	O1R	18.14

Table 23 (continued)

Graph	$k = 2$		$k = 3$		$k = 4$		$k = 5$	
	Ordering	Running time, sec	Ordering	Running time, sec	Ordering	Running time, sec	Ordering	Running time, sec
ln2-sanchis-100-50	C1R	0.20	C3	0.13	C3R	10.17	O1R	43.99
ln2-sanchis-100-60	C1R	0.06	O1	1.80	O1R	139.38	O1R	6553.74
ln2-sanchis-100-70	C1R	0.06	C1	16.34	O1R	>10800	O1R	>10800
ln2-sanchis-100-80	C1R	0.08	C2R	10.94	O1R	89.89	O1R	>10800
ln2-sanchis-100-90	O1R	0.03	O1R	0.05	O1	0.05	DR	0.05
ln2-sanchis-1000-40	O1R	>10800	O1R	>10800	C2	>10800	C1	>10800
ln2-sanchis-1000-50	O1R	>10800	O1R	>10800	O1R	>10800	C1	>10800
ln2-sanchis-1000-60	O1R	>10800	O1R	>10800	C3R	>10800	O1	>10800
ln2-sanchis-1000-70	O1R	>10800	O1R	>10800	O1	>10800	O1	>10800
ln2-sanchis-1000-80	O1R	>10800	O1	>10800	O1R	>10800	C1	>10800
ln2-sanchis-1000-90	O1	>10800	O1	>10800	O1	>10800	O1	>10800
ln2-sanchis-200-40	C1	46.89	O1R	>10800	O1R	>10800	C4	>10800
ln2-sanchis-200-50	O1R	9946.19	O1R	>10800	C1	>10800	C4R	>10800
ln2-sanchis-200-60	O1	393.28	C3	1028.47	C2R	>10800	O1	>10800
ln2-sanchis-200-70	C3	>10800	O1R	1051.50	O1	>10800	O1R	>10800
ln2-sanchis-200-80	C1R	365.91	O1R	>10800	O1R	>10800	O1R	>10800
ln2-sanchis-200-90	O1	0.11	O1R	8.39	O1	1116.05	O1R	>10800
ln2-sanchis-300-40	O1R	>10800	O1R	>10800	C4	>10800	C1	>10800
ln2-sanchis-300-50	O1R	>10800	O1R	>10800	C2	>10800	C1R	>10800
ln2-sanchis-300-60	O1R	>10800	O1R	>10800	C2R	>10800	O1	>10800
ln2-sanchis-300-70	O1R	>10800	C1R	>10800	O1	>10800	O1R	>10800
ln2-sanchis-300-80	O1R	>10800	O1	>10800	O1R	>10800	C4	>10800
ln2-sanchis-300-90	O1R	>10800	O1R	>10800	O1	>10800	O1	>10800
ln2-sanchis-400-40	O1R	>10800	O1R	>10800	C5	>10800	C1	>10800
ln2-sanchis-400-50	O1R	>10800	C3	>10800	C1R	>10800	C3R	>10800
ln2-sanchis-400-60	O1R	>10800	C3	>10800	C1	>10800	O1	>10800
ln2-sanchis-400-70	C2	>10800	C1	>10800	O1	>10800	O1R	>10800
ln2-sanchis-400-80	C1	>10800	O1	>10800	O1	>10800	O1R	>10800
ln2-sanchis-400-90	O1R	>10800	O1	>10800	O1	>10800	O1	>10800
ln2-sanchis-500-40	O1R	>10800	O1R	>10800	C5	>10800	C2	>10800
ln2-sanchis-500-50	O1R	>10800	C3	>10800	C1	>10800	C2R	>10800

Table 23 (continued)

Graph	$k = 2$		$k = 3$		$k = 4$		$k = 5$	
	Ordering	Running time, sec	Ordering	Running time, sec	Ordering	Running time, sec	Ordering	Running time, sec
ln2-sanchis-500-60	O1R	>10800	O1R	>10800	C1R	>10800	O1	>10800
ln2-sanchis-500-70	O1R	>10800	O1R	>10800	O1	>10800	O1R	>10800
ln2-sanchis-500-80	O1R	>10800	O1	>10800	O1R	>10800	O1R	>10800
ln2-sanchis-500-90	O1	>10800	O1R	>10800	O1	>10800	O1	>10800
ln2-sanchis-600-40	O1R	>10800	O1R	>10800	C3	>10800	C2	>10800
ln2-sanchis-600-50	O1R	>10800	C4	>10800	C1	>10800	C1	>10800
ln2-sanchis-600-60	O1R	>10800	C1	>10800	C1	>10800	O1	>10800
ln2-sanchis-600-70	O1R	>10800	O1R	>10800	O1	>10800	C1	>10800
ln2-sanchis-600-80	O1R	>10800	O1	>10800	O1R	>10800	C1	>10800
ln2-sanchis-600-90	O1	>10800	O1R	>10800	O1	>10800	O1	>10800
ln2-sanchis-700-40	O1R	>10800	O1R	>10800	C4	>10800	C1	>10800
ln2-sanchis-700-50	O1R	>10800	C3	>10800	C1	>10800	C1	>10800
ln2-sanchis-700-60	O1R	>10800	O1R	>10800	C1	>10800	O1	>10800
ln2-sanchis-700-70	O1R	>10800	O1R	>10800	O1	>10800	O1R	>10800
ln2-sanchis-700-80	O1R	>10800	O1	>10800	O1R	>10800	O1R	>10800
ln2-sanchis-700-90	O1	>10800	O1	>10800	O1	>10800	O1	>10800
ln2-sanchis-800-40	O1R	>10800	O1R	>10800	C1	>10800	C3	>10800
ln2-sanchis-800-50	O1R	>10800	O1R	>10800	C4	>10800	C1R	>10800
ln2-sanchis-800-60	O1R	>10800	O1R	>10800	C2R	>10800	O1	>10800
ln2-sanchis-800-70	O1R	>10800	C1	>10800	O1	>10800	C1	>10800
ln2-sanchis-800-80	O1R	>10800	O1	>10800	O1R	>10800	C1	>10800
ln2-sanchis-800-90	O1	>10800	O1R	>10800	O1	>10800	O1	>10800
ln2-sanchis-900-40	O1R	>10800	O1R	>10800	C3	>10800	C1	>10800
ln2-sanchis-900-50	O1R	>10800	O1R	>10800	C1	>10800	C1	>10800
ln2-sanchis-900-60	C3	>10800	C1	>10800	C1	>10800	C1	>10800
ln2-sanchis-900-70	O1R	>10800	O1R	>10800	O1	>10800	C1	>10800
ln2-sanchis-900-80	O1R	>10800	O1R	>10800	O1R	>10800	O1R	>10800
ln2-sanchis-900-90	O1R	>10800	O1R	>10800	O1	>10800	O1	>10800
mulsol.i.1.col	C3	0.08	C1	0.09	O2	0.31	O2	0.72
n5-sanchis-100-40	O1	0.03	C2R	0.06	C2R	0.58	C2R	20.22
n5-sanchis-100-50	O1	0.05	C1	0.16	C1	10.17	O1R	112.07

Table 23 (continued)

Graph	$k = 2$		$k = 3$		$k = 4$		$k = 5$	
	Ordering	Running time, sec	Ordering	Running time, sec	Ordering	Running time, sec	Ordering	Running time, sec
n5-sanchis-100-60	O1	0.06	C1R	1.34	O1R	179.99	O1R	7151.72
n5-sanchis-100-70	O1	1.53	O1	47.27	O1R	6592.18	O1R	>10800
n5-sanchis-100-80	O1	26.80	O1R	9103.81	O1R	>10800	C1	>10800
n5-sanchis-100-90	O1	185.33	O1	>10800	C1	>10800	C1	>10800
n5-sanchis-1000-40	C2R	0.66	W4	11.39	O1R	>10800	O1R	>10800
n5-sanchis-1000-50	C3R	1.72	C5R	47.06	O1R	>10800	O1R	>10800
n5-sanchis-1000-60	C3R	14.89	O1R	>10800	O1R	>10800	O1	>10800
n5-sanchis-1000-70	C1	>10800	O1R	>10800	O1R	>10800	O1R	>10800
n5-sanchis-1000-80	O1R	>10800	O1	>10800	O1R	>10800	O1R	>10800
n5-sanchis-1000-90	O1R	>10800	O1R	>10800	O1	>10800	O1	>10800
n5-sanchis-200-40	C1	0.08	C2	0.16	C2	1.36	C3	33.69
n5-sanchis-200-50	O1	0.11	O1	0.64	O1	19.22	O1R	>10800
n5-sanchis-200-60	C1R	0.16	C1R	12.97	O1R	>10800	O1R	>10800
n5-sanchis-200-70	D	12.42	O1R	>10800	O1R	>10800	O1R	>10800
n5-sanchis-200-80	O1R	>10800	O1R	>10800	O1R	>10800	O1R	>10800
n5-sanchis-200-90	O1R	>10800	O1R	>10800	O1R	>10800	C2R	>10800
n5-sanchis-300-40	C1R	0.13	C2	0.39	DR	12.73	O1R	>10800
n5-sanchis-300-50	C1	0.13	C2R	2.70	C2	73.05	O1R	>10800
n5-sanchis-300-60	C1	0.22	C1	14.42	O1R	>10800	C1R	>10800
n5-sanchis-300-70	C2R	17.86	O1R	>10800	O1R	>10800	O1R	>10800
n5-sanchis-300-80	O1R	>10800	O1R	>10800	O1R	>10800	C3	>10800
n5-sanchis-300-90	O1R	>10800	O1R	>10800	O1	>10800	O1	>10800
n5-sanchis-400-40	C1R	0.17	C5	0.88	C5	12.06	O1R	>10800
n5-sanchis-400-50	C1	0.20	C3R	14.91	C5R	130.29	O1R	>10800
n5-sanchis-400-60	C1R	0.83	C1	118.13	O1R	>10800	O1	>10800
n5-sanchis-400-70	C5R	31.11	O1R	>10800	O1	>10800	O1R	>10800
n5-sanchis-400-80	O1R	>10800	O1R	>10800	O1R	>10800	O1R	>10800
n5-sanchis-400-90	O1R	>10800	O1R	>10800	O1	>10800	O1	>10800
n5-sanchis-500-40	DR	0.23	DR	1.11	DR	20.38	O1R	>10800
n5-sanchis-500-50	C2R	0.31	W1	12.66	O1R	>10800	O1R	>10800
n5-sanchis-500-60	C1	0.69	C1	45.28	O1R	>10800	C1R	>10800

Table 23 (continued)

Graph	$k = 2$		$k = 3$		$k = 4$		$k = 5$	
	Ordering	Running time, sec	Ordering	Running time, sec	Ordering	Running time, sec	Ordering	Running time, sec
n5-sanchis-500-70	C3R	38.47	O1R	>10800	O1R	>10800	O1R	>10800
n5-sanchis-500-80	O1R	>10800	O1R	>10800	O1R	>10800	O1R	>10800
n5-sanchis-500-90	O1R	>10800	O1R	>10800	O1	>10800	O1	>10800
n5-sanchis-600-40	C1R	0.27	C5R	1.61	C5R	30.36	O1R	>10800
n5-sanchis-600-50	C1R	0.36	C1	12.64	O1R	>10800	O1R	>10800
n5-sanchis-600-60	C1	1.28	C2R	122.68	O1R	>10800	O1R	>10800
n5-sanchis-600-70	C1	24.41	O1R	>10800	O1	>10800	O1R	>10800
n5-sanchis-600-80	O1R	>10800	O1R	>10800	O1R	>10800	O1R	>10800
n5-sanchis-600-90	O1R	>10800	O1R	>10800	O1	>10800	O1	>10800
n5-sanchis-700-40	C1R	0.44	C3R	10.23	C2R	518.10	O1R	>10800
n5-sanchis-700-50	D	1.06	C1	25.58	O1R	>10800	O1R	>10800
n5-sanchis-700-60	C1	1.44	O1R	>10800	O1R	>10800	C5	>10800
n5-sanchis-700-70	C1R	223.47	O1R	>10800	O1	>10800	O1R	>10800
n5-sanchis-700-80	O1R	>10800	O1R	>10800	O1R	>10800	O1R	>10800
n5-sanchis-700-90	O1R	>10800	O1R	>10800	C2R	>10800	O1	>10800
n5-sanchis-800-40	C1R	0.48	C1R	10.47	C5	>10800	O1R	>10800
n5-sanchis-800-50	C1R	0.88	C1	22.88	O1R	>10800	O1R	>10800
n5-sanchis-800-60	W2	17.09	C2	>10800	O1R	>10800	O1	>10800
n5-sanchis-800-70	C1	63.00	O1R	>10800	O1R	>10800	O1R	>10800
n5-sanchis-800-80	O1R	>10800	O1R	>10800	O1R	>10800	C1	>10800
n5-sanchis-800-90	O1R	>10800	O1R	>10800	O1	>10800	O1	>10800
n5-sanchis-900-40	C1R	0.48	C5	11.17	O1R	>10800	O1R	>10800
n5-sanchis-900-50	C1R	0.73	C5R	18.31	O1R	>10800	O1R	>10800
n5-sanchis-900-60	O2	9.80	C4	>10800	O1R	>10800	C3R	>10800
n5-sanchis-900-70	C1R	127.91	O1R	>10800	C1R	>10800	O1R	>10800
n5-sanchis-900-80	O1R	>10800	O1R	>10800	O1R	>10800	O1R	>10800
n5-sanchis-900-90	O1R	>10800	O1R	>10800	O1	>10800	O1	>10800
p_hat300-1.clq	O1R	0.48	O1R	41.74	O1R	3041.44	O1R	>10800
p_hat300-2.clq	C3	2097.06	C1	>10800	O1R	>10800	O1R	>10800
p_hat300-3.clq	O1R	>10800	O1R	>10800	O1R	>10800	O1R	>10800
p_hat500-1.clq	O1	10.53	O1R	1640.52	O1R	>10800	C2	>10800

Table 23 (continued)

Graph	$k = 2$		$k = 3$		$k = 4$		$k = 5$	
	Ordering	Running time, sec	Ordering	Running time, sec	Ordering	Running time, sec	Ordering	Running time, sec
p_hat500-2.clq	O1R	>10800	O1R	>10800	O1R	>10800	O1R	>10800
p_hat500-3.clq	O1R	>10800	O1R	>10800	O1R	>10800	O1	>10800
r100.5	O1	0.08	O1R	2.22	O1R	94.11	O1R	3305.26
r200.5	C2	10.00	O1R	597.51	O1R	>10800	O1R	>10800
r300.5	O1R	111.60	O1R	>10800	O1R	>10800	O1R	>10800
r400.5	O1R	1173.21	O1R	>10800	O1R	>10800	O1R	>10800
r500.5	O1R	4402.76	O1R	>10800	O1R	>10800	O1R	>10800
san200_0.7_1.clq	C2	>10800	C2	>10800	C2	>10800	C2	>10800
san200_0.7_2.clq	C1	>10800	C1	>10800	C4	>10800	C1	>10800
san200_0.9_1.clq	C1	>10800	O1	0.19	O1R	19.92	O1	2403.20
san200_0.9_2.clq	O1R	>10800	O1	17.14	O1R	>10800	O1	>10800
san200_0.9_3.clq	C5R	>10800	O1R	>10800	O1R	>10800	C2	>10800
san400_0.5_1.clq	O1R	>10800	C1	>10800	C1	>10800	C3	>10800
san400_0.7_1.clq	O1R	>10800	C1	>10800	C1	>10800	C3R	>10800
san400_0.7_2.clq	C1	>10800	C5R	>10800	C4R	>10800	C3	>10800
san400_0.7_3.clq	O1R	>10800	O1R	>10800	C3	>10800	C4	>10800
san400_0.9_1.clq	C4	>10800	O1R	>10800	O1R	>10800	O1R	>10800
sanr200_0.7.clq	O1R	1411.91	O1R	>10800	O1R	>10800	O1R	>10800
sanr200_0.9.clq	O1R	>10800	O1R	>10800	O1R	>10800	C2R	>10800
sanr400_0.5.clq	O1R	1276.39	O1R	>10800	O1R	>10800	O1R	>10800
sanr400_0.7.clq	O1R	>10800	O1R	>10800	O1R	>10800	O1R	>10800
school1.col	C5	>10800	C5	>10800	C3	>10800	O1R	>10800
school1_nsh.col	O1R	84.11	C1	24.45	C1	2381.73	O1R	>10800

Table 24 Real-life networks information

Graph	Nodes	Edges	Density, %	Maximum k-plex size				
				$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
Erdos97-1	472	1314	1.18	7	8	8	9	11
Erdos97-2	5488	8972	0.06	7	8	9	11	12
Erdos98-1	485	1381	1.18	7	8	9	11	12
Erdos98-2	5822	9505	0.06	7	8	9	11	12
Erdos99-1	492	1417	1.17	7	8	8	10	11
Erdos99-2	6100	9939	0.05	8	8	9	11	12
HPylori	1570	1401	0.11	3	5	6	7	8
SCerevisiae	2112	2203	0.10	6	6	7	7	8
Football2005	119	633	9.02	9	10	11	12	12
AA.net	90	331	8.26	7	9	10	11	12
CO.net	73	156	5.94	5	6	7	8	9
DL.net	104	330	6.16	7	9	10	11	12
NW.net	113	317	5.01	7	8	9	10	11
UA.net	80	258	8.16	7	9	10	11	11
US.net	63	282	14.44	7	8	10	11	12
WN.net	63	793	40.60	14	18	20	22	23
WNc.net	63	733	37.53	13	18	19	21	22
All7.net	162	1944	14.91	20	25	28	30	32
SkyTeam.net	140	760	7.81	11	12	15	17	19
USAir97	332	2126	3.87	22	24	26	28	29

Table 25 Real-life network instances running time

Graph	$k = 2$		$k = 3$		$k = 4$		$k = 5$	
	Ordering	Running time, sec	Ordering	Running time, sec	Ordering	Running time, sec	Ordering	Running time, sec
Erdos97-1	W1R	0.16	D	0.16	D	0.22	D	0.30
Erdos97-2	D	2.03	D	3.78	D	60.64	D	1932.11
Erdos98-1	D	0.17	D	0.19	D	0.28	W1R	1.36
Erdos98-2	D	2.19	D	4.44	D	83.47	D	2920.34
Erdos99-1	D	0.17	D	0.19	D	0.19	D	0.28
Erdos99-2	D	2.31	D	4.73	D	90.41	D	3172.94
HPylori	D	0.53	D	0.55	W1R	0.78	D	2.36
SCerevisiae	D	0.69	D	0.72	D	0.80	D	1.69
Football2005	O1R	0.05	O1R	0.05	WR	0.05	D	0.05
AA.net	O1R	0.03	W1R	0.03	D	0.09	WR	0.92
CO.net	O1R	0.03	O1R	0.05	O2R	0.08	O2R	0.59
DL.net	O1R	0.03	DR	0.05	O2R	0.16	O2R	1.78
NW.net	C	0.05	W1R	0.05	WR	0.23	WR	3.00
UA.net	CR	0.03	DR	0.03	D	0.06	D	0.59
US.net	C1	0.01	C1	0.03	O1R	0.05	O2R	0.23
WN.net	C1R	0.03	DR	0.02	C1	0.03	O1R	0.13
WNc.net	C	0.02	O1R	0.03	O1R	0.05	C1	0.14
All7.net	O1	0.08	O1	0.19	O1	2.59	DR	12.05
SkyTeam.net	O1R	0.05	WR	0.06	W1R	0.14	D	0.67
USAir97	D	0.14	D	0.19	D	0.77	D	7.25

Table 26 Running time comparson with McClosky's algorithm

	Runtime, sec					
	$k = 2$		$k = 3$		$k = 4$	
Graph	M	T	M	T	M	T
brock200_1	-	-	-	-	-	-
brock200_2	64	4.74	-	446.68	-	-
brock200_4	-	398.65	-	-	-	-
brock400_2	-	-	-	-	-	-
brock400_4	-	-	-	-	-	-
c-fat200-1	0	0.06	0	0.08	18	0.08
c-fat200-2	0	0.08	0	0.08	3	0.11
c-fat200-5	0	0.08	0	0.08	1	0.13
c-fat500-1	0	0.16	8	0.17	1234	0.20
c-fat500-2	0	0.19	2	0.17	92	0.25
c-fat500-5	0	0.17	1	0.19	8	0.27
c-fat500-10	0	0.19	0	0.22	4	0.33
gen400_p0.9_55	-	-	-	-	-	-
gen400_p0.9_65	-	-	-	-	-	-
gen400_p0.9_75	-	-	-	-	-	-
hamming6-2	0	0.03	1	0.14	951	131.86
hamming6-4	0	0.02	0	0.03	1	0.16
hamming8-2	1	0.13	-	1500.83	-	-
hamming8-4	58	7.45	-	-	-	-
johnson8-2-4	0	0.01	0	0.01	0	0.05
johnson8-4-4	0	0.05	35	4.92	-	768.80
johnson16-2-4	-	3124.10	-	-	-	-
johnson32-2-4	-	-	-	-	-	-
keller4	913	55.45	-	-	-	-
MANN_a9	0	0.01	2	0.03	141	6.23
MANN_a27	-	-	-	0.24	-	-
p_hat300-1	5	0.48	416	41.74	-	3041.44
p_hat300-2	-	2097.06	-	-	-	-
p_hat300-3	-	-	-	-	-	-
san200_0.7_2	-	-	-	-	-	-
san200_0.9_1	-	-	964	0.19	-	19.92
san200_0.9_2	-	-	-	17.14	-	-

Table 27 Running time comparson with Balasundaram's algorithms

	Runtime, sec		
Graph	BC-MIS	BC-C2PLX	T
MANN_a27.clq	>10800	>10800	>10800
MANN_a9.clq	0.262	0.289	0.01
brock200_1.clq	>10800	>10800	>10800
c-fat200-1.clq	25.891	212.239	0.06
c-fat200-2.clq	24.235	7636.49	0.08
c-fat200-5.clq	90.564	5006.05	0.08
c-fat500-1.clq	1263.81	9587.21	0.16
c-fat500-10.clq	>10800	>10800	0.19
c-fat500-2.clq	2985.04	>10800	0.19
c-fat500-5.clq	10142.8	>10800	0.17
hamming6-2.clq	0.421	1.686	0.03
hamming6-4.clq	4.609	6.767	0.02
hamming8-2.clq	>10800	>10800	0.13
hamming8-4.clq	>10800	>10800	7.45
johnson16-2-4.clq	>10800	>10800	3124.10
johnson32-2-4.clq	>10800	>10800	>10800
johnson8-2-4.clq	1.952	1.171	0.01
johnson8-4-4.clq	1951.87	3283.15	0.05
keller4.clq	>10800	>10800	55.45
ln2-sanchis-100-40	11.92	124.92	0.08
ln2-sanchis-100-50	29.13	307.09	0.20
ln2-sanchis-100-60	29.23	479.00	0.06
ln2-sanchis-100-70	19.36	505.87	0.06
ln2-sanchis-100-80	1.67	56.47	0.08
ln2-sanchis-100-90	0.01	0.02	0.03
ln2-sanchis-200-40	684.53	>10800	46.89
ln2-sanchis-200-50	2467.22	>10800	9946.19
ln2-sanchis-200-60	7920.65	>10800	393.28
ln2-sanchis-200-70	>10800	>10800	>10800
ln2-sanchis-200-80	>10800	>10800	365.91
ln2-sanchis-200-90	1.09	1.08	0.11
ln2-sanchis-300-40	9050.57	-	>10800
ln2-sanchis-300-50	>10800	>10800	>10800

Table 27 (continued)

Graph	Runtime, sec		
	BC-MIS	BC-C2PLX	T
ln2-sanchis-300-60	>10800	-	>10800
ln2-sanchis-300-90	>10800	>10800	>10800
ln2-sanchis-400-40	>10800	-	>10800
ln2-sanchis-400-90	>10800	-	>10800
ln2-sanchis-500-40	>10800	-	>10800
n5-sanchis-100-40	2.02	18.20	0.03
n5-sanchis-100-50	6.31	107.10	0.05
n5-sanchis-100-60	21.91	426.46	0.06
n5-sanchis-100-70	325.18	4109.14	1.53
n5-sanchis-100-80	>10800	>10800	26.80
n5-sanchis-100-90	291.59	>10800	185.33
n5-sanchis-1000-40	>10800	-	0.66
n5-sanchis-200-40	16.75	667.42	0.08
n5-sanchis-200-50	92.91	5418.70	0.11
n5-sanchis-200-60	179.19	>10800	0.16
n5-sanchis-200-70	2025.52	>10800	12.42
n5-sanchis-200-80	>10800	-	>10800
n5-sanchis-200-90	>10800	-	>10800
n5-sanchis-300-40	84.27	7453.67	0.13
n5-sanchis-300-50	539.76	>10800	0.13
n5-sanchis-300-60	1115.40	-	0.22
n5-sanchis-300-70	>10800	-	17.86
n5-sanchis-400-40	298.60	>10800	0.17
n5-sanchis-400-50	2106.16	>10800	0.20
n5-sanchis-400-60	4392.91	-	0.83
n5-sanchis-400-70	>10800	-	31.11
n5-sanchis-500-40	855.32	-	0.23
n5-sanchis-500-50	5675.32	-	0.31
n5-sanchis-500-60	>10800	-	0.69
n5-sanchis-600-40	1761.98	-	0.27
n5-sanchis-600-50	>10800	-	0.36
n5-sanchis-600-60	>10800	-	1.28
n5-sanchis-700-40	3495.51	-	0.44

Table 27 (continued)

Graph	Runtime, sec		
	BC-MIS	BC-C2PLX	T
n5-sanchis-700-50	>10800	-	1.06
n5-sanchis-700-60	>10800	-	1.44
n5-sanchis-800-40	6491.40	-	0.48
n5-sanchis-900-40	10627.00	-	0.48

Table 28 Numerical results for the market graphs for one-year period

				Maximum clique			Maximum 2-plex			
Graph	Nodes	Edges	Density, %	Weight	Order	Running time, sec	Weight	Order	Running time, sec	Ordering
market-1990	840	65467	18.58	4601	7	0.00	6542	6	0.58	D
market-1991	1262	220338	27.69	13506	8	1.00	16518	8	10.77	C1R
market-1992	1409	342536	34.53	6150	8	1.00	6354	10	34.27	C2R
market-1993	1780	525225	33.17	7629	10	2.00	8720	9	13.19	C2R
market-1994	2066	718034	33.66	9333	8	3.00	13637	8	71.72	C2R
market-1995	2249	849059	33.59	10246	10	12.00	12958	9	850.15	C2R
market-1996	2642	969952	27.80	14506	9	10.00	18104	9	1012.05	C1R
market-1997	2937	913459	21.19	9844	9	5.00	14103	8	44.86	C2R
market-1998	3181	1252979	24.77	2055	7	1.00	2545	6	6.99	D
market-1999	3448	1826182	30.73	4826	8	4.00	5470	8	19.94	C2R
market-2000	3931	2014359	26.08	9310	8	22.00	9614	8	859.81	W2R
market-2001	4196	1953603	22.20	3920	8	13.00	4304	10	420.08	W2
market-2002	4360	2225668	23.42	4100	9	34.00	5101	10	2135.82	C2R
market-2003	4606	2426222	22.88	5127	8	52.00	5845	9	7416.76	C1R
market-2004	4986	2858289	23.00	16523	9	20.00	18841	8	472.62	C1R
market-2005	5469	3690795	24.68	13407	9	42.00	15693	9	1905.94	C1R
market-2006	5972	3612892	20.26	2587	8	7.00	3539	10	56.88	C1R

VITA

Svyatoslav Trukhanov received his Bachelor and Master of Science degrees in Computer Science from Kyiv National Taras Shevchenko University in Ukraine. He entered the doctoral program in Industrial and Systems Engineering Department of Texas A&M University in January 2004 and received his Doctor of Philosophy degree in August 2008. His research interests are in combinatorial optimization, with data-mining applications in financial, social, biological and communication networks. He will join the team of Server and Tools Business Division at Microsoft Corporation in October 2008.

Mr. Trukhanov may be reached at his work address:

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

This document was typeset in LaTeX by Svyatoslav Trukhanov.